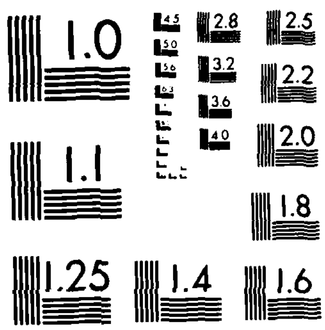
AD-A151 927 ESTIMATION OF LAUNCH VEHICLE PERFORMANCE PARAMETERS(U) 1/2
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL
OF ENGINEERING D A VALLADO DEC 84 AFIT/GA/AA/84D-10

UNCLASSIFIED F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GA/AA/84D-10

(1)

ESTIMATION OF LAUNCH VEHICLE

PERFORMANCE PARAMETERS

THESIS

David A. Vallado
Captain USAF

AFIT/GA/AA/84D-10

Approved for Public release: distribution unlimited

AFIT/GA/AA/84D-10

ESTIMATION OF LAUNCH VEHICLE PERFORMANCE PARAMETERS

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in partial fulfillment of the

requirements for the degree of

Master of Science

DAVID A. VALLADO, M.S., B.S.

Captain USAF

December 1984

Approved for Public release: distribution unlimited

ACKNOWLEDGEMENTS

I would like to express my thanks to my advisor Dr. Wiesel whose help was invaluable in completing this thesis. He provided tremendous help, motivation and insight during the course of study.

Next, the VAX-11/780 deserves it's due credit. Finally, I would like to thank AFIT for it was in this program that I met my wife Laura.

TABLE OF CONTENTS

Table of Contents Continued

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The estimation of launch vehicle performance parameters was explored through the use of a Bayes filter. The main emphasis was to use an eight state model that would include the vehicle position and velocity vectors, the vehicle exhaust velocity, and the ratio of the mass flow rate to the initial mass. A primary objective was to be able to observe these quantities through the staging events, where the last two elements of the state would be changing very rapidly. The results indicated that indeed the staging event was observable. However, as would be expected, the data processed at the exact time of staging included errors which diminished as the filter processed more data. A fading memory was added in an attempt to improve the filters performance in the area of the staging event. This proved to be marginally successful as several Bayes loop iterations had to be performed to notice the effect of the fading memory addition. Care was taken to show each step of the filter development and its checkout. Several numerical tables are presented including the input and output data.

I INTRODUCTION

The specific problem that will be examined in this paper is the estimation of launch vehicle performance parameters from either a ground based or a space based sensor system. We can easily understand the importance of this since, from a military point of view, it is a necessity that the United States be able to obtain the launch vehicle characteristics of the weapons of hostile countries. This is essential in the planning of our strategic policies and is needed to determine where the emphasis should be placed in trying to develop or improve existing U.S. systems. The individual parameters must be known to some tolerance so an adequate comparison can be made with respect to the U.S. systems. Since these countries do not publish any data relating to their launch vehicle systems, we must rely on estimation systems which use observation data to determine these parameters. The data can take many forms since it can be supplied from radar systems that are ground or space-based, or any other form of detection. However in all cases, the basic form of the estimator is essentially the same.

The problem scenario is developed as follows. Upon the launch of an ICBM from a hostile country, our sensor systems respond by first detecting that a launch has indeed occured, and the subsequent tracking procedure is then carried out. The utility of the estimator, the development of which will

be examined in this paper, is to take this subsequent tracking data and estimate the vehicle performance parameters. Upon successful target recognition, data transmittal, and estimation calculation, some of the ICBM's performance characteristics will be known.

The following diagram shows the general geometry for an orbiting sensor.

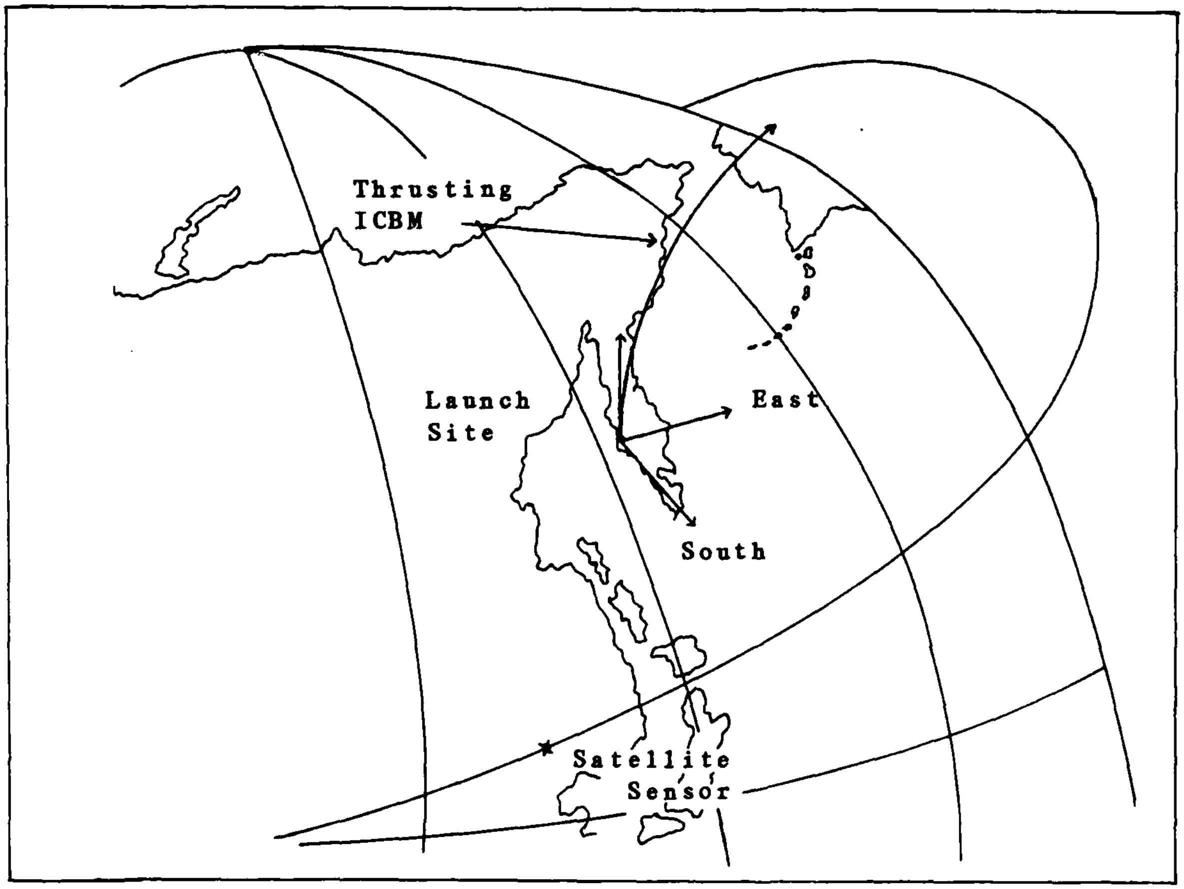


Figure 1 Orbital Sensor Geometry

Note that the initial conditions, to be discussed later, will be the launch site of the ICBM, either on the ground or

in the silo. Recognizing the physical limitations in acquiring the target, some time will elapse before the tracking procedure is fully functioning, and the trajectory portion that is actually observed will be only a subset of the entire flight.

The situation with a land based sensor is very similar, with a few distinctions. The basic situation is shown below.

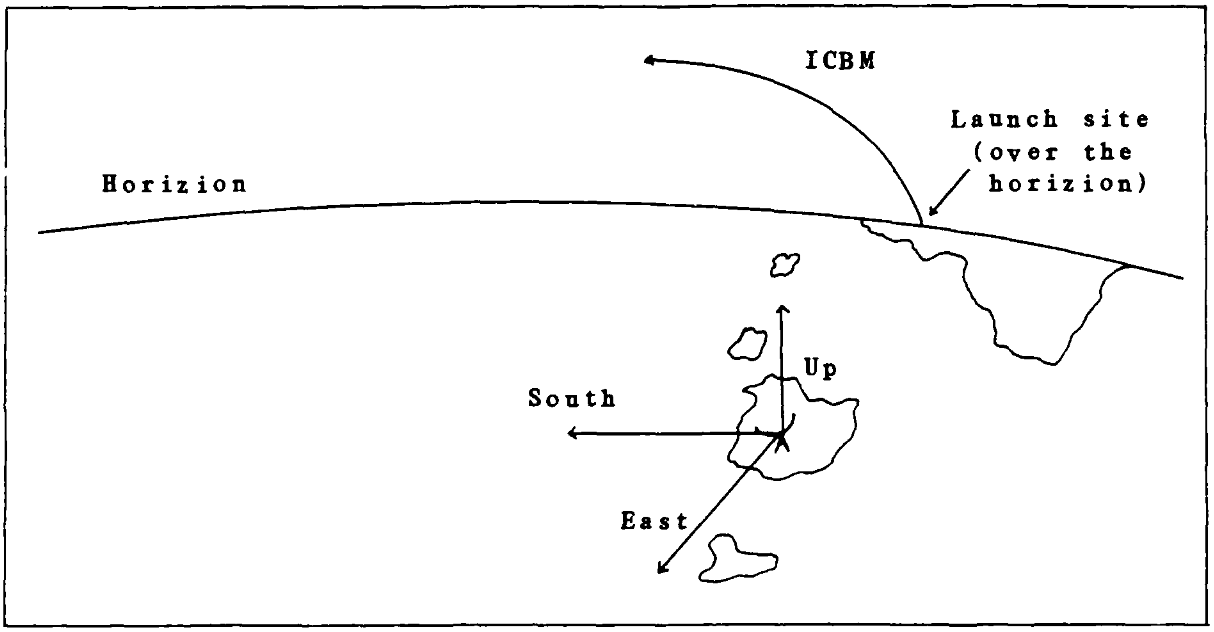


Figure 2. Land Based Sensor Geometry

Added difficulty appears in target acquistion since, although the site may be closer, instead of looking down on the target launch, the sensor is forced to discriminate as the vehicle rises over the horizion. In addition, depending on the launch site, the vehicle may not be in view until a significant part of it's trajectory has been flown. This would only serve to complicate the tracking procedure, and

where

x, y, z are the 3 components of position
$\dot{x}$, $\dot{y}$, $\dot{z}$ are the 3 components of velocity
V_e is the exhaust velocity
$\dot{M}$ is the ratio $\dot{m}$ over initial mass

The two-body equation is non-linear, so to move the state through time, it is differentiated and written in a general form of the equation of motion as :

$$
{}^{d}/_{dt}\,(\bar{x}(t)) = F\,(\bar{x}(t),t) \qquad (3-2)
$$

where $\bar{x}(t)$ is the state vector at each time.

Notice that this is simply a different expression for equation 2-5.

Using the equations of motion that were developed in the last section, the F vector is found as follows:

$$
F = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{V}e \\ \dot{M} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ -x^{\mu}/_{r}3 \quad + \; a\; {}^{\dot{x}}/_{v} \\ -y^{\mu}/_{r}3 \quad + \; a\; {}^{\dot{y}}/_{v} \\ -z^{\mu}/_{r}3 \quad + \; a\; {}^{\dot{z}}/_{v} \\ 0 \\ 0 \end{bmatrix} \qquad (3-3)
$$

where $\bar{a} = V_e\frac{\dot{M}}{1-\dot{M}t}$

Note here that the V_e and the $\dot{M}$ are assumed to be constant. In this program, variations in exhaust velocity and mass ratio are handled through the white Gaussian noise of the

III FILTER DEVELOPMENT

Matrix Equations

The filter will be receiving sampled input data from the sensors. In order to process this data, a Bayes filter estimator was used. The Bayes filter was chosen since the problem dynamics are basically deterministic. Equations of motion can be written specifically for the vehicle, and since it is desired to evaluate each stage's performance, several data segments will need to be processed to obtain the results. The bayes filter uses sequential data segments and produces current estimates of the state and covariance, thereby lending itself to the specific problem of observing each of the stage's performance.

To start, define the state vector ($\bar{x}$) describing the state of the vehicle. Recalling the launch vehicle equations of motion:

$$\bar{x} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ Ve \\ M \end{bmatrix} \qquad (3\text{-}1)$$

The partial of range, azimuth, and elevation then becomes an identity matrix since the data matrix (G) contains only the variables (range, azimuth, elevation) and no functions of these quantities. The 'noisy' data is then formed as:

$$\begin{bmatrix} \text{range} \\ \text{azimuth} \\ \text{elevation} \end{bmatrix}_{\text{noisy}} = \begin{bmatrix} \text{range} \\ \text{azimuth} \\ \text{elevation} \end{bmatrix}_{\text{perfect}} + \delta \begin{pmatrix} \text{range} \\ \text{azimuth} \\ \text{elevation} \end{pmatrix} \qquad (2\text{-}23)$$

The next task is to develop the filter which will implement the dynamics formulated in this chapter.

A range of vehicles was chosen to ensure that the filter would operate, given a wide variety of possible input trajectories.

Runs made with the previous information produced actual results. Next the programs in Appendix A were run containing the option to calculate noisy data so that the program could simulate incorrect measurements from the sensors. The result here was to have the data files written with the random errors included in each measuerment of range, azimuth and elevation. Basically the approach taken was to assume that the errors would occur randomly as per a Gaussian distribution. To obtain the difference between measurements with and without the noise included (designated d( )), let:

$$d\begin{pmatrix}\text{range}\\ \text{azimuth}\\ \text{elevation}\end{pmatrix} = G_{au}\,\sigma\begin{pmatrix}\text{range}\\ \text{azimuth}\\ \text{elevation}\end{pmatrix} \qquad (2\text{-}20)$$

where

G_{au} represents a gaussian function whose mean = 0 and standard deviation is ± 1

σ(range, azimuth, elevation)isthe defined accuracy of each measurement

The σ(range, azimuth, elevation) accuracies were input as follows:

$$\sigma\begin{pmatrix}\text{range}\\ \text{azimuth}\\ \text{elevation}\end{pmatrix} = \begin{bmatrix}.00001\ \text{DU}\\ .001\ \text{deg}\\ .001\ \text{deg}\end{bmatrix} \qquad \cong \qquad 64\text{m} \qquad (2\text{-}21)$$

Then, using partial derivatives the deviations are:

$$\delta\begin{pmatrix}\text{range}\\ \text{azimuth}\\ \text{elevation}\end{pmatrix} = G_{au}\begin{bmatrix}.00001\\ .001\\ .001\end{bmatrix}\ \partial\begin{pmatrix}\text{range}\\ \text{azimuth}\\ \text{elevation}\end{pmatrix} \qquad (2\text{-}22)$$

Table 1. Launch Vehicle data

Quantity	Stg I	Stg II	Stg III	Stg IV
Titan IIIB				
I_{sp} sec	256	317	292 (Agena)	
m_o lb	305970	73816	14676	
F lb	434900	102300	16000	
Titan IIID				
I_{sp} sec	301	317	444	284
m_o lb	307500	73670	36122	2721
F lb	523000	102300	30000	15000
Thor LV-2F				
I_{sp} sec	251	290		
m_o lb	106092	1743.7		
F lb	170000	10000		

Using equation (2-19) the parameters are calculated which are needed to enable the correct calculation of the A matrix. The A matrix is simply the assemblage of the equations of variation. The values are as follows.

Table 2. Launch Vehicle Performance Parameters

Stg	Quantity	TitanIIIB	TitanIIID	ThorLV-2F
I	V_e	.317298	.373693	.311618
	M	4.479637	4.551364	5.142138
II	V_e	.393557	.393557	.360036
	M	3.521417	3.528396	15.928772
III	V_e	.362519	.551228	
	M	3.007333	1.506670	
IV	V_e		.352587	
	M		15.634947	

where

V_e is in DU/TU
M is non dimensional

$$\bar{\rho}_{SEZ} = \begin{bmatrix} \_\_\hat{S}\_\_ \\ \_\_\hat{E}\_\_ \\ \hat{Z} \end{bmatrix} \bar{\rho}_{IJK} \qquad (2\text{-}17)$$

Then, recalling Figure 4, the data is assembled as:

$$\begin{aligned} \text{range} &= \rho \\ \text{azimuth} &= \tan^{-1} ({}^{y} / {}_{x}) \qquad (2\text{-}18) \\ \text{elevation} &= \tan^{-1} ({}^{z} / \sqrt{x^2 + y^2}) \end{aligned}$$

This result is general in that the mechanization of finding the unit vectors SEZ is the same for both the radar site and the orbiting sensor. The only difference is the initial calculation of $\bar{r}_s$.

Truth Model Development

Programming the equations of motion and numerically integrating them provide the numerical integration and truth model data. Appendix A lists the programs which were used to generate the data. In order to look at the launch vehicle in particular however, something must be known about their characteristics, specifically, the exhaust velocity V_e, and the mass ratio, $M = \dot{m}$ over initial mass. This data was obtained for a few different missiles from the U.S. Space Launch Systems document, published by the Navy, Reference 5. Recalling from basic propulsion (Sutton Reference 4) that:

$$\dot{m} = {}^{F}/{}_{V_e} \quad \text{and} \quad V_e = I_{sp}\, g \quad \text{and} \quad M = {}^{\dot{m}}/{}_{m_o} \qquad (2\text{-}19)$$

only I_{sp}, m_o, and F need to be obtained. The values are as follows.

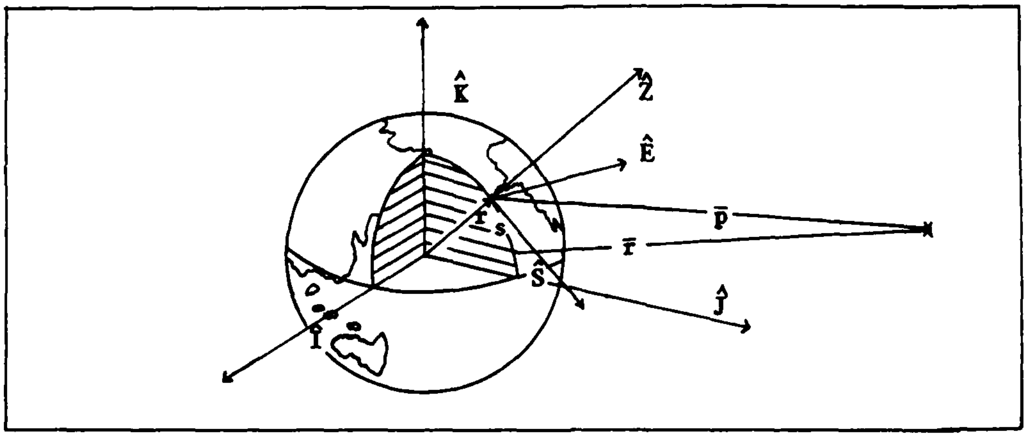


Figure 5. Observation Geometry

$$\hat{Z} = \bar{r}_s \, / \, |\bar{r}_s| \tag{2-11}$$

the east unit vector then is

$$\hat{E} = \hat{k} \times \hat{Z} \, / \, |\hat{k} \times \hat{Z}| \tag{2-12}$$

and:

$$\hat{S} = \hat{E} \times \hat{Z} \, / \, |\hat{E} \times \hat{Z}| \tag{2-13}$$

The transformation matrix is then formed as

$$\begin{bmatrix} I \\ J \\ K \end{bmatrix} = \begin{bmatrix} \hat{S} & | & \hat{E} & | & \hat{Z} \end{bmatrix} \begin{bmatrix} S \\ E \\ Z \end{bmatrix} \tag{2-14}$$

Recalling that the inverse of an orthogonal basis vector is the same as the transpose, this is also:

$$\begin{bmatrix} S \\ E \\ Z \end{bmatrix} = \begin{bmatrix} -\hat{S}- \\ -\hat{E}- \\ \hat{Z} \end{bmatrix} \cdot \begin{bmatrix} I \\ J \\ K \end{bmatrix} \tag{2-15}$$

To complete the process of finding range, azimuth and elevation, the launch vehicle's position vector is computed as:

$$\bar{\rho}_{IJK} = \bar{r} - \bar{r}_s \quad (IJK) \tag{2-16}$$

transforming to SEZ:

process.) The position vector that is calculated from the orbit elements represents the $\bar{r}_s$ of the orbiting sensor at that time.

With the site vector for both cases, calculations for the range, azimuth, elevation, and a local coordinate system need to be developed. As will be seen later, the calculation of the data matrix G and the observation matrix H (partial of G with respect to the state vector) is much simplified by the proper choice of coordinate systems. For this reason, the SEZ system was chosen. (Reference 1) With this type of system, range, azimuth and elevation are depicted as follows:

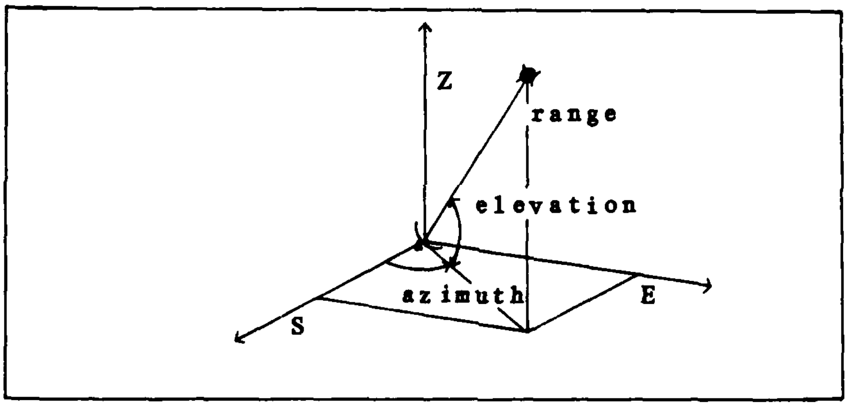


Figure 4. Radar Site geometry

Notice that azimuth is defined as being measured from the South unit vector, rather than the North. Since all of the data is input in the IJK frame, an orthogonal set of unit vectors, SEZ, must be found so that a transformation can be set up. To obtain these unit vectors, Figure 5 is used. The local vertical unit vector is obtained by making the site a unit vector so:

The coordinate system for a land based radar site is shown in Figure 3. Notice that given only latitude, local sidereal time and elevation, the site position vector is obtained.

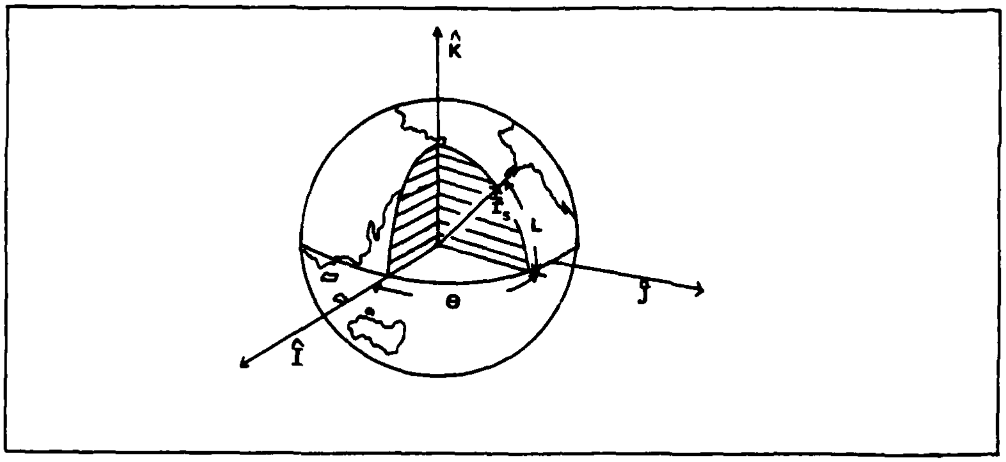


Figure 3. Radar Site Coordinate System

Looking now at an orbiting sensor, the orbit of the sensor must be known. The 5 classical orbit elements are input (Reference 1), and position and velocity vectors can be calculated. From these, the mean motion is calculated as:

$$n = \sqrt{\mu / a^3} \qquad (2\text{-}9)$$

where

n = mean motion
a = semi major axis

The mean anomaly M is then calculated as:

$$M = n (t - t_o) \qquad (2\text{-}10)$$

where $(t - t_o)$ = the time in dayspastthe initial time

Position and velocity vectors can then be calculated. (See Appendix A for a more complete description of the $\bar{r}$ and $\bar{v}$

examined an orbiting sensor looking down upon the ICBM trajectory. The observation relationships can be considered identical once the position vector of the observer (radar site, or orbiting sensor) are known.

For a radar site, the position vector can be determined given latitude, longitude, elevation, and universal time. The first step is to calculate the local sidereal time for the site.

$$\theta = \theta_g + \lambda_e \tag{2-6}$$

where

θ = Local sidereal time
θ_g = greenwich sidereal time
λ_e = longitude of the site

The Greenwich Sidereal Time is then calculated as:

$$\theta_g = \theta_{go} + 1.0027379093\ (\ t - t_o\ )2\pi \tag{2-7}$$

where

θ_g = Greenwich Sidereal Time (deg)
θ_{go} = the value in deg on 1 Jan 1984, (98.85481°)
$(t - t_o)$= the time in days past the initial time

The position vector of the site can then be calculated as:

$$\bar{r}_s = \begin{bmatrix} h \cos(L) \cos(\theta) \\ h \cos(L) \sin(\theta) \\ h \sin(L) \end{bmatrix} \tag{2-8}$$

where

h = the distance from center of the earth to the site
$\bar{r}_s$ = the site vector
L = latitude of the site
θ = local sidereal time

that would be considered.

The vehicle will also undergo an acceleration due to thrusting during the propulsive phase of the flight which is equal to:

$$\bar{a} = V_e \frac{\dot{m}}{m_o - \dot{m}t} \qquad (2\text{-}3)$$

where

$\bar{a}$ = vehicle acceleration due to thrust
$\dot{m}$ = mass flow rate
m_o = initial mass
t = time
V_e = Vehicle exhaust velocity

But since absolute masses are not observable from the trajectory data, let $M = \dot{m}/_{m_o}$, and:

$$\bar{a} = V_e \frac{M}{1-Mt} \qquad (2\text{-}4)$$

To obtain the total vehicle acceleration, equations (2-1) and (2-4) are added to get:

$$\ddot{\bar{r}} = -\bar{r}\mu/_{r^3} + V_e \frac{M}{1-Mt} \qquad (2\text{-}5)$$

where the $\ddot{\bar{r}}$ denotes the total vehicle acceleration.

These equations constitute the equations of motion for the launch vehicle when they are numerically integrated. The next task is to develop the observation relationships.

Observation Relationships

Two cases were considered for the observer. The first case considered a radar site observing the trajectory of an ICBM as it could be seen above the horizon. The second case

II DYNAMICS FORMULATION

Equations of Motion

For the specific problem, the equations of motion for the trajectory of the launch vehicle must be generated. Numerically integrating these equations on the computer, will simulate the data that the radar sites would be observing and providing to the sensor system.

The underlying assumption of this work is the spherical earth model and the use of the two body equation of motion (Reference 1), and Newton's Law that the mass times the acceleration is equal to the sum of the forces. In general, the two body equation of motion is written as:

$$\ddot{\bar{r}} + \bar{r}\, \mu / r^3 = 0 \tag{2-1}$$

where

$\ddot{\bar{r}}$ = vehicle acceleration
$\bar{r}$ = radius vector from center of Earth to vehicle
r = radius vector magnitude
μ = gravitational parameter defined by:

$$\mu = GM \tag{2-2}$$

where

G = Gravitatrional potential
M = Mass of the Earth

Notice that only gravitational forces are considered since in this paper, all other external forces are assumed to be zero. The assumption is made since the acceleration due to thrust is several orders of magnitude larger than the other forces

the use of radar data in conjunction with the infared data so that range, azimuth and elevation measurements will be available for computation. In addition, a different local coordinate system is adopted to make the computations easier in the derivation of the various observation relationships. The analysis starts with a specific look at the problem and what data is avialable. The dynamics are then formulated, yielding the equations of motion and the filter algorithm that will produce tangible results. In order to assess the results of the filter algorithm, a truth model was developed which simulated the data for each parameter that was modeled. Specifically, a reference thrust profile was developed through the computer programs contained in the appendices, and this was used as the 'true' profile. The simulation then proceeds to try to observe this quantity, and the performance of the estimator can thus be observed.

make the target acquisition process slower.

Previous work was done using infared sensors to determine the position and velocity vectors and vehicle acceleration for the launch vehicle (Reference 3). The method used azimuth and elevation measurements and concluded that a 7-state filter could solve for vehicle acceleration. One of the problems with this particular approach was the extreme complexity of the data and observation relationships. This made for extremely difficult checkout and computer coding. Gross's presentation (Reference 2), provided additional work centered on a space-based infared sensor system in an attempt to get additional data from the orbiting sensor. The main emphasis here centered on a Bayes filter to observe exhaust velocity and vehicle mass, and the same filter to observe vehicle acceleration. The simulation did not yield significant results for the first case, but was able to observe the vehicle acceleration. The main result seemed to conclude that a fading memory differential corrector might be useful and that additional input data could change the poor results in the expanded estimation system. As before however, the observation relationships were extremely involved.

The majority of this paper will be based on trying to extend the Bayes filter analysis already started in an attempt to develop a filter that will provide as much data as possible. The primary difference with this attempt will be

observations. The dynamics model could incorporate very complex expressions to represent the variations, but this was not done in this paper.

The previous relations establish the laws that will govern the motion of the launch vehicle. The next problem is to determine how to correct the estimate of the state from the input data (coming from the radar sites or satellites), which the estimator will be proccessing. To accomplish this, a vector is found through the observation relationships which were developed in the previous chapter, and the following operations are performed.

In order to estimate the state, a nominal trajectory is assumed as a function of time($\bar{x}(t)$), with initial conditions, and the true trajectory is written as:

$$\bar{x}(t) = \bar{x}_0(t) + \delta\bar{x}(t) \qquad (3\text{-}4)$$

where

$\bar{x}(t)$ = the true solution
$\delta\bar{x}(t)$ = the difference between the nominal and true trajectory

Differentiating yields:

$$\dot{\bar{x}}(t) = \dot{\bar{x}}_0(t) + \delta\dot{\bar{x}}(t) \qquad (3\text{-}5)$$

and adding to Equation (3-2):

$$\dot{\bar{x}}_0(t) + \delta\dot{\bar{x}}(t) = F(\bar{x}_0(t) + \delta\bar{x}(t), t) \qquad (3\text{-}6)$$

To solve this we expand the right hand side of Equation 3-6 with Taylor's theorem, and obtain:

$$\dot{\bar{x}}_o(t) + \delta\dot{\bar{x}}(t) = F(\bar{x}_o(t),t) + A(t)\Big|_{\bar{x}_o(t)} \delta\bar{x}(t)$$
$$+ {}^1/_2 \, \nabla_x (\nabla_x \delta) \Big|_{\bar{x}_o(t)} (\delta\bar{x}(t))^2 + \text{H.O.T.} \qquad (3\text{-}7)$$

where $A(t) = \partial F/\partial\bar{x}$ (the A matrix is derived in Appendix D)

Now assuming that $\delta\bar{x}$ is small, Equation (3-7) becomes:

$$\dot{\bar{x}}_o(t) = F(\bar{x}_o(t),t) \qquad (3\text{-}8)$$

Ignoring higher order terms in Equation (3-7) and subtracting Equation (3-8), we obtain:

$$\delta\dot{\bar{x}}(t) = A(t)\Big|_{\bar{x}_o(t)} \delta\bar{x}(t) \qquad (3\text{-}9)$$

Recalling the idea of state transition matrices, the state variations are moved through time as follows:

$$\delta\bar{x}(t) = \phi(t,t_o)\,\delta\bar{x}(t_o) \qquad (3\text{-}10)$$

where $\phi(t,t_o)$ is the state transition matrix

ϕ is then calculated from:

$$\phi(t,t_o) = A(t)\Big|_{\bar{x}_o(t)} \phi(t,t_o) \qquad (3\text{-}11)$$

The state transition matrix definition also prescribes the initial conditions as:

$$\phi(t_o,t_o) = I \qquad (3\text{-}12)$$

The preceding equations will enable the calculation of the state vector, propagation through time, and the estimation of the errors from the true trajectory as a function of time.

The next part of the problem is to process the data that will be coming in from the sensors. The predicted data for each observation is given by:

$$\bar{z}(t_i) = G(\bar{x}(t_i),t_i) \qquad (3\text{-}13)$$

Evaluating this at the initial time, the initial conditions become:

$$\bar{z}_0(t_i) = G(\bar{x}_0(t_i), t_i) \qquad (3\text{-}14)$$

Also, knowing that there will be some difference between this and the true measurment, let:

$$\bar{z}(t_i) = G(\bar{x}_0(t_i) + \delta\bar{x}(t_i), t_i) \qquad (3\text{-}15)$$

where $\delta\bar{x}(t_i)$ is the difference from the true data

This equation can be expanded in a Taylor series as follows:

$$\bar{z}(t_i) = G(\bar{x}_0(t_i), t_i) + \left.\frac{\partial G(\bar{x}_0(t_i), t_i)}{\partial \bar{x}(t_i)}\right|_{\bar{x}_0(t_i)} \delta\bar{x}(t_i) + \text{H. O. T.} \qquad (3\text{-}16)$$

Subtracting this 'true' relation from the calculated relation gives the residual of the observation:

$$\begin{aligned} \bar{r}(t_i) &= \bar{z}(t_i) - \bar{z}_0(t_i) \\ &= \left.{}^{\partial G}/_{\partial \bar{x}}\right|_{\bar{x}_0\,(t)} \delta\bar{x}(t_i) \\ &= H(\bar{x}_0(t_i), t_i)\delta\bar{x}(t_i) \end{aligned} \qquad (3\text{-}17)$$

Note that here, as before, the higher order terms have been ignored.

In the previous chapters, observation relationships were developed. The data vector G consists of the range, azimuth and elevation.

$$[G] = \begin{bmatrix} \text{range} \\ \text{azimuth} \\ \text{elevation} \end{bmatrix} \qquad (3\text{-}18)$$

The H matrix (observation relation) is simply the partial derivative of G with respect to the state vector, so:

$$[H] = \partial G / \partial \bar{x} \qquad (3\text{-}19)$$

Using the observation relationships that were developed in the last chapters:

$$\begin{array}{lcl} \text{range} & & \sqrt{x^2 + y^2 + z^2} \\ \text{azimuth} & = & \tan^{-1}\left(y / x \right) \\ \text{elevation} & & \tan^{-1}\left(z / \sqrt{x^2 + y^2} \right) \end{array} \qquad (3\text{-}20)$$

Since only x, y, and z appear in the G matrix, only the first 3 x 3 block of the H matrix will have non zero elements. Using the following definition then:

$$d/dx \tan^{-1} u = 1/_{1+u} \; (du/dx) \qquad (3\text{-}21)$$

The first 3x3 of the H matrix is then:

$$H = \begin{bmatrix} \dfrac{x}{(x^2+y^2+z^2)^{1/2}} & \dfrac{y}{(x^2+y^2+z^2)^{1/2}} & \dfrac{z}{(x^2+y^2+z^2)^{1/2}} \\ \dfrac{-y/x^2}{1+(y/x)^2} & \dfrac{1/x}{1+(y/x)^2} & 0 \\ \dfrac{-xz/(x^2+y^2)^{3/2}}{1+ z^2/(x^2+y^2)} & \dfrac{-yz/(x^2+y^2)^{3/2}}{1+ z^2/(x^2+y^2)} & \dfrac{1/(x^2+y^2)^{1/2}}{1+ z^2/(x^2+y^2)} \end{bmatrix}$$

(3-22)

The final step is to move the residuals to a single epoch time. Using the state transition matrix which was developed before:

$$\begin{aligned} \bar{r}(t_i) &= H(\bar{x}_0(t_i), t_i)\ \phi\ (t_i, t_0)\ \delta\bar{x}(t_0) \\ &= T(t_i)\ \delta\bar{x}(t_0) \end{aligned} \qquad (3\text{-}23)$$

With the residuals calculated, the necessary matrices are available and the filter can be derived.

Least Squares Filter Development

Summarizing the data already obtained, the state vector is given by:

$$\dot{\bar{x}} = F(\bar{x}, t) \tag{3-24}$$

deviations of the state vector are known as:

$$\delta\bar{x}(t) = \phi\ (t, t_o)\ \delta\bar{x}(t_o) \tag{3-25}$$

The observation realtionships were developed as G, and the residual data was:

$$\bar{r}(t_i) = T(t_i)\ \delta\bar{x}(t_o) \tag{3-26}$$

The sensors will not input perfect data, and the covariance matrix Q tells us how accurate the data is (for each measurment of range, azimuth and elevation), and how each quantity affects the other. (See chapter 2 for numerical values.) The residual vector, including this error is written as:

$$\bar{r}(t_i) = T(t_i)\ \delta\bar{x}(t_o) + \bar{e}(t_i) \tag{3-27}$$

To calculate this error, rearrange as:

$$\bar{e}(t_i) = \bar{r}(t_i) - T(t_i)\ \delta\bar{x}(t_o) \tag{3-28}$$

Assuming that for each observation, a random errors in range, azimuth or elevation are uncorrelated with each other, a data covariance matrix (Q) defined which contains the information about the accuaracy of the measurments. Using Gaussian error

statistics, the probability density function for the error vector $\bar{e}(t_i)$ is:

$$P(\bar{e}) = (2\pi)^{-N/2} \; [\; Q \;]^{-1/2} \; \exp(^{-J}/_{2}) \qquad (3\text{-}29)$$

where

N = number of measurements
Q = data covariance matrix
J = $\bar{e}^T Q^{-1} \bar{e}$ (a scalar) wieghted least squares function

Using the principle of maximum likelihood, J is minimized to make P a maximum. (P is a maximum when the residual errors e are the smallest) Thus:

$$\frac{\partial J}{\partial x} = \frac{\partial}{\partial \bar{x}} (\bar{e}^T Q^{-1} \bar{e}) = 0 \qquad (3\text{-}30)$$

Now substituting into J as:

$$J = (\bar{r} - T\,\delta\bar{x})^T\, Q^{-1}\, (\bar{r} - T\,\delta\bar{x}) \qquad (3\text{-}31)$$

$$= \bar{r}^T Q^{-1} \bar{r} - \bar{r}^T Q^{-1} T \delta\bar{x} - \delta\bar{x} T^T Q^{-1} \bar{r} + \delta\bar{x}^T T^T Q^{-1} T \delta\bar{x}$$

Note here that the functional dependence on time is intentionally left out to enhance clarity. Thus, Equation (3-30) becomes:

$$0 = -(\bar{r}^T Q^{-1} T)^T - T^T Q^{-1} \bar{r} + (\delta\bar{x}^T T^T Q^{-1} T)^T + T^T Q^{-1} T \delta\bar{x}$$

$$= -2T^T Q^{-1} \bar{r} + 2T^T Q^{-1} T \delta\bar{x} \qquad (3\text{-}32)$$

and solving for $\delta\bar{x}$:

$$\delta\bar{x} = (T^T Q^{-1} T)^{-1}\; T^T Q^{-1} \bar{r} \qquad (3\text{-}33)$$

This result is valid when the trajectories are very close. Iteration is needed to get the trajectories to have a very small difference.

Next, the covariance is to be calculated. In general,

the $\delta\bar{x}$ can be written as:

$$\delta\bar{x} = W\bar{r} \tag{3-34}$$

where $W = (T^TQ^{-1}T)^{-1}\ T^TQ^{-1}$

The covariance of $\bar{x}$ which produced this error, given $\bar{x}(t_o)$ at the epoch time (assuming $\delta\bar{x}$ as zero mean) is:

$$P_{\bar{x}}(t_o) = E(\delta\bar{x},\delta\bar{x}^T) \tag{3-35}$$

where $\delta\bar{x}$ is $W\bar{r} = \bar{x} - \bar{x}_o$

Substituting and recognizing that W can be *calculated (the* assumption of deterministic dynamics for the problem), it is pulled out of the expectation operator:

$$P_{\bar{x}}(t_o) = W\ E(\bar{r}\bar{r}^T)W^T \tag{3-36}$$

But recall that $E(\bar{r}\bar{r}^T)$ is defined as the covariance matrix Q, where $\bar{r}$ is the zero mean, thus:

$$P_{\bar{x}}(t_o) = W\ Q\ W^T \tag{3-37}$$

Expanding,

$$\begin{aligned} P_{\bar{x}}(t_o) &= (T^TQ^{-1}T)^{-1}\ T^TQ^{-1}\ Q[(T^TQ^{-1}T)^{-1}\ T^TQ^{-1}]^T \\ &= (T^TQ^{-1}T)^{-1}\ T^TQ^{-1}T\ (T^TQ^{-1}T)^{-1} \\ &= (T^TQ^{-1}T)^{-1} \end{aligned} \tag{3-38}$$

The final step which is needed is to define when the estimator has reached convergence. Under perfect conditions, the δx will converge to zero. However, it is sufficient to stop the iteration when the state corrections are all less than the square root of their individual covariance values. Knowing this, the algorithm for the least squares filter can

be summarized and written as shown in Appendix H.

Bayes Filter Development

The process of changing to a Bayes filter is relatively easy since the only real difference is that the Bayes filter will process segments of data in a least squares mode, and then update it's state and covariance matrix to some time in the future. The major changes are detailed below.

Again recalling the Least Squares development, the matrices that will change are as follows, realizing that the previous estimate and matrices can now be treated as 'data':

$$T = \begin{bmatrix} I \\ \hline T_n \end{bmatrix} \qquad Q = \left[\begin{array}{c|c} P(-) & 0 \\ \hline 0 & Q_n \end{array}\right]$$

$$\bar{r} = \begin{bmatrix} \bar{x}(-) - \bar{x}_{ref} \\ \hline Z_n - G(\bar{x})_n \end{bmatrix} \tag{3-39}$$

where the subscript n denotes the new portion of data

Then:

$$P^{-1}(+) = (P^{-1}(-) \quad T_n{}^T Q_n{}^{-1}) \begin{bmatrix} I \\ \hline T_n \end{bmatrix}$$

$$= P^{-1}(-) + T_n{}^T Q_n{}^{-1} T_n \tag{3-40}$$

and the corrections are:

$$\delta\bar{x}(t_o) = P(+) T^T Q^{-1} \bar{r} \tag{3-41}$$

$$= P(+) \, (P^{-1}(-) \quad T_n{}^T Q_n{}^{-1}) \begin{bmatrix} \bar{x}(-) - \bar{x}_{ref} \\ \hline \bar{r}_n \end{bmatrix}$$

and finally:

$$\delta\bar{x}(t_o) = P(+) \, (P^{-1}(-)(\bar{x}(-) - \bar{x}_{ref}) + T_n{}^T Q_n{}^{-1} \bar{r}_n) \tag{3-42}$$

One additional device must be added to the least squares developement to incorporate the fading memory effect. The object here is to have the filter retain full memory about components which do not change drastically during the flight, (position and velocity vectors) and to retain little memory of those elements which will change rapidly at certain times, (exhaust velocity and mass ratio). The staging event is the primary cause of the changes. The fading memory filtering is accomplished by multiplying the covariance matrix by a matrix of scalar β's after convergence has been acheived.

$$P^{-1}(-) = \beta P^{-1}(-) \beta^{T} \tag{3-43}$$

Values of β range from 0 to 1. When $\beta = 0$, the filter does not retain memory about the previous states, and when $\beta = 1$, the filter retains all previous data.

The Summary for the Bayes Filter Algorithm is shown in Appendix I.

IV TESTING

Computer Program Development

The computer programs were developed by simply coding the equations and formulas which were developed in the previous sections. The programs also relied on basis programs which were demonstrated during the Modern Methods of Orbit Determination Course. (Reference 7) The appendices give a brief description of each of the programs, with their inputs and outputs.

To ensure the programs were correct, a succession of checks were run to determine that each stage of the program was indeed functional.

The first check was of the numerical integrator . A program was written which accomplished this by numerically integrating an orbit, once around. Appendix A details the inputs and outputs, and the procedures used. It was noted that the number of steps was crucial in solving the problem. A step size that was too large could not be used with a high altitude orbit, or an elliptical orbit. The starting point for the integration was also very important since at perigee, the spacecraft is moving faster, thereby requiring smaller step sizes. Table 3 lists numerical integration results for an orbit giving position (DU) and velocity (DU/TU) vectors through one revolution. If the numerical integrator were perfect, and there were no roundoff errors, the first and

Table 3. Satellite Orbit, Numerical Integration Data

a	e	i	omega	argp	nuo	m	period	# it
2.500	.000	.785	.000	.000	.000	.000	333.973	0

the specific mech energy and ang momentum are

x	y	z	xdot	ydot	zdot
-.20000000000		1.58113883008			
2.50000000000	.00000000000	.00000000000	.00000000000	.44721359551	.44721359549
2.48028675328	.22155994842	.22155994841	-.07926769692	.44368718273	.44368718271
2.42145790282	.43962576408	.43962576406	-.15728529490	.43316355804	.43316355802
2.32444121472	.65075841883	.65075841880	-.23282240982	.41580868527	.41580868526
2.19076670011	.85162822456	.85162822452	-.30468777626	.39189626115	.39189626113
2.02254248594	1.03906734445	1.03906734441	-.37174803446	.36180339888	.36180339887
1.82242156856	1.21011975162	1.21011975156	-.43294560408	.32600468088	.32600468087
1.59355997439	1.36208784741	1.36208784735	-.48731536289	.28506467432	.28506467431
1.33956698747	1.49257500418	1.49257500411	-.53399986733	.23962902756	.23962902755
1.06444822895	1.59952336151	1.59952336144	-.57226287486	.19041428788	.19041428787
-.20000000001		1.58113883006			
.77254248598	1.68124627992	1.68124627984	-.60150095500	.13819660113	.13819660113
.46845328652	1.73645494017	1.73645494009	-.62125300577	.08379947143	.08379947143
.15697629889	1.76427866871	1.76427866863	-.63120752551	.02808077401	.02808077401
-.15697629875	1.76427866871	1.76427866863	-.63120752551	-.02808077400	-.02808077399
-.46845328638	1.73645494018	1.73645494010	-.62125300577	-.08379947142	-.08379947142
-.77254248585	1.68124627993	1.68124627985	-.60150095501	-.13819660112	-.13819660112
-1.06444822881	1.59952336152	1.59952336145	-.57226287487	-.19041428787	-.19041428786
-1.33956698734	1.49257500419	1.49257500412	-.53399986734	-.23962902755	-.23962902754
-1.59355997426	1.36208784742	1.36208784736	-.48731536291	-.28506467432	-.28506467431
-1.82242156844	1.21011975163	1.21011975158	-.43294560409	-.32600468089	-.32600468087
-.20000000001		1.58113883004			
-2.02254248582	1.03906734446	1.03906734442	-.37174803446	-.36180339889	-.36180339887
-2.19076669999	.85162822456	.85162822453	-.30468777626	-.39189626116	-.39189626114
-2.32444121459	.65075841882	.65075841880	-.23282240982	-.41580868528	-.41580868527
-2.42145790269	.43962576408	.43962576406	-.15728529489	-.43316355805	-.43316355803
-2.48028675314	.22155994841	.22155994840	-.07926769690	-.44368718275	-.44368718273
-2.49999999984	.00000000000	.00000000000	.00000000002	-.44721359552	-.44721359550
-2.48028675312	-.22155994841	-.22155994840	.07926769694	-.44368718275	-.44368718273
-2.42145790264	-.43962576408	-.43962576406	.15728529494	-.43316355805	-.43316355803
-2.32444121452	-.65075841882	-.65075841880	.23282240987	-.41580868528	-.41580868526
-2.19076669988	-.85162822456	-.85162822452	.30468777632	-.39189626115	-.39189626114
-.20000000002		1.58113883002			
-2.02254248569	-1.03906734445	-1.03906734441	.37174803452	-.36180339888	-.36180339886
-1.82242156827	-1.21011975161	-1.21011975156	.43294560415	-.32600468087	-.32600468085
-1.59355997406	-1.36208784739	-1.36208784733	.48731536297	-.28506467430	-.28506467428
-1.33956698711	-1.49257500415	-1.49257500408	.53399986741	-.23962902752	-.23962902751
-1.06444822855	-1.59952336146	-1.59952336138	.57226287494	-.19041428783	-.19041428782
-.77254248555	-1.68124627984	-1.68124627976	.60150095507	-.13819660107	-.13819660106
-.46845328605	-1.73645494005	-1.73645493998	.62125300583	-.08379947136	-.08379947135
-.15697629839	-1.76427866855	-1.76427866847	.63120752556	-.02808077392	-.02808077392
.15697629927	-1.76427866850	-1.76427866843	.63120752555	.02808077410	.02808077409
.46845328691	-1.73645493992	-1.73645493984	.62125300579	.08379947153	.08379947153
-.20000000002		1.58113883000			
.77254248638	-1.68124627962	-1.68124627954	.60150095500	.13819660124	.13819660123
1.06444822934	-1.59952336115	-1.59952336108	.57226287484	.19041428799	.19041428798
1.33956698784	-1.49257500376	-1.49257500369	.53399986728	.23962902768	.23962902766
1.59355997472	-1.36208784693	-1.36208784687	.48731536281	.28506467444	.28506467443
1.82242156885	-1.21011975108	-1.21011975102	.43294560397	.32600468100	.32600468099
2.02254248616	-1.03906734385	-1.03906734381	.37174803431	.36180339900	.36180339898
2.19076670025	-.85162822390	-.85162822386	.30468777609	.39189626126	.39189626124
2.32444121476	-.65075841812	-.65075841809	.23282240962	.41580868537	.41580868535
2.42145790274	-.43962576333	-.43962576331	.15728529466	.43316355811	.43316355809
2.48028675308	-.22155994764	-.22155994763	.07926769666	.44368718279	.44368718277
-.20000000003		1.58113882998			

x	y	z	xdot	ydot	zdot
2.49999999966	.00000000079	.00000000079	-.00000000028	.44721359554	.44721359552

last state vectors would be identical. The very small, 10^{-9} errors, represent the imperfections.

Table 4 lists sample ouput for the launch trajectory. The data is for a Titan IIIB,launched from a site at 53.7° N, 158.2° E. The initial velocity was 200 ft/sec straight up (local coordinate system), and to displace the velocity so that the vehicle would execute a gravity turn, this initial value was perturbed by .06 DU/TU. In a gravity turn there is a pre-programmed displacement to perturb the vehicle's direction so that gravity will cause the vehicle to fall over and reach burnout in the correct orientation. The output contains the position and velocity magnitudes, γ , and the time, V_e and M. Note that with a .06 displacement, at the end of data, the ICBM is virtually horizontal ($\gamma = 80°$), and in orbit. This is easily changed by altering the amount of the displacement.

The next step was to check the A matrix. Appendix D lists the A matrix, and the program which accomplished the check. Given an initial state vector, the A matrix was calculated . Then, each element of the input state was perturbed, and the A matrix was calculated by columns as:

$$A_{ij} = \frac{F_i(\bar{x} + \delta, t) - F_i(\bar{x}\ , \ t)}{\delta} \qquad (4\text{-}1)$$

Table 5 lists the results for a trial case. Note that each state was perturbed by about 1 x 10^{-4}, and the observed delta

Table 4. ICBM Launch Trajectory, Numerical Integration Data

The initial state vector for the missile is

x	y	z	xdot
-.4579093757416e+00	-.5625717569603e+00	.6883545756938e+00	-.3531037967384e-02
ydot	**zdot**	**Ve**	**M**
-.4338112164634e-02	.5626533647914e-02	.3172982551730e+00	.4479637558632e+01

the initial time is .0020 TU

r (km)	v (ft/sec)	gamma (deg)	time (sec)	ve (DU/TU)	m
6378.430588	263.814633	2.8953242033	5.6136237488	.3172982552	4.4796375586
6378.789127	326.359237	4.4846479830	9.6136237488	.3172982552	4.4796375586
6379.225596	393.765854	6.4144524540	13.6136237488	.3172982552	4.4796375586
6379.744871	466.471967	8.6493455674	17.6136237488	.3172982552	4.4796375586
6380.351603	544.993952	11.1451502446	21.6136237488	.3172982552	4.4796375586
6381.050078	629.919834	13.8525358355	25.6136237488	.3172982552	4.4796375586
6381.844101	721.896054	16.7200663601	29.6136237488	.3172982552	4.4796375586
6382.736897	821.610423	19.6967954942	33.6136237488	.3172982552	4.4796375586
6383.731040	929.773906	22.7344241792	37.6136237488	.3172982552	4.4796375586
6384.828431	1047.103788	25.7889803070	41.6136237488	.3172982552	4.4796375586
6386.030301	1174.310352	28.8219788486	45.6136237488	.3172982552	4.4796375586
6387.337255	1312.088445	31.8010569378	49.6136237488	.3172982552	4.4796375586
6388.749345	1461.114507	34.7001278663	53.6136237488	.3172982552	4.4796375586
6390.266165	1622.048931	37.4991401939	57.6136237488	.3172982552	4.4796375586
6391.886952	1795.543147	40.1835516549	61.6136237488	.3172982552	4.4796375586
6393.610703	1982.250598	42.7436302088	65.6136237488	.3172982552	4.4796375586
6395.436292	2182.840797	45.1736810688	69.6136237488	.3172982552	4.4796375586
6397.362579	2398.015809	47.4712761083	73.6136237488	.3172982552	4.4796375586
6399.388524	2628.528764	49.6365374388	77.6136237488	.3172982552	4.4796375586
6401.494541	2823.542549	51.6896474704	81.6136237488	.3929044800	3.5272654570
6403.655508	3027.078349	53.6529187642	85.6136237488	.3929044800	3.5272654570
6405.868128	3242.066483	55.5259659308	89.6136237488	.3929044800	3.5272654570
6408.129193	3468.884445	57.3093457140	93.6136237488	.3929044800	3.5272654570
6410.435582	3707.938547	59.0043861052	97.6136237488	.3929044800	3.5272654570
6412.784312	3959.672608	60.6130028733	101.6136237488	.3929044800	3.5272654570
6415.172571	4224.576523	62.1375427256	105.6136237488	.3929044800	3.5272654570
6417.597755	4503.195472	63.5806516542	109.6136237488	.3929044800	3.5272654570
6420.057503	4796.139981	64.9451664714	113.6136237488	.3929044800	3.5272654570
6422.549729	5104.097117	66.2340269751	117.6136237488	.3929044800	3.5272654570
6425.072654	5427.843164	67.4502059825	121.6136237488	.3929044800	3.5272654570
6427.624844	5768.258228	68.5966544919	125.6136237488	.3929044800	3.5272654570
6430.205238	6126.343318	69.6762594049	129.6136237488	.3929044800	3.5272654570
6432.813194	6503.240598	70.6918114876	133.6136237488	.3929044800	3.5272654570
6435.448525	6900.257717	71.6459815223	137.6136237488	.3929044800	3.5272654570
6438.111553	7318.897373	72.5413028780	141.6136237488	.3929044800	3.5272654570
6440.803161	7760.893661	73.3801589795	145.6136237488	.3929044800	3.5272654570
6443.524858	8228.257251	74.1647743763	149.6136237488	.3929044800	3.5272654570
6446.278862	8723.332209	74.8972082936	153.6136237488	.3929044800	3.5272654570
6449.068189	9248.868313	75.5793496879	157.6136237488	.3929044800	3.5272654570
6451.896772	9808.114288	76.2129129190	161.6136237488	.3929044800	3.5272654570
6454.769599	10404.939722	76.7994331915	165.6136237488	.3929044800	3.5272654570
6457.692897	11043.996945	77.3402608925	169.6136237488	.3929044800	3.5272654570
6460.674355	11730.939759	77.8365538476	173.6136237488	.3929044800	3.5272654570
6463.723417	12472.724757	78.2892662990	177.6136237488	.3929044800	3.5272654570
6466.851662	13278.035739	78.6991330153	181.6136237488	.3929044800	3.5272654570
6470.073319	14157.896975	79.0666462800	185.6136237488	.3929044800	3.5272654570
6473.405962	15126.585985	79.3920223679	189.6136237488	.3929044800	3.5272654570
6476.871492	16203.040581	79.6751521371	193.6136237488	.3929044800	3.5272654570
6480.497557	17413.120586	79.9155267793	197.6136237488	.3929044800	3.5272654570
6484.319703	18793.434422	80.1121229189	201.6136237488	.3929044800	3.5272654570

end of data run

Table 5. A ϕ H Matrix Data from Checkout

```
the A matrix check data is as follows for
the initial state vector y of
-.132611000e+00 -.576970000e+00  .005928000e+00 -.102300000e-02
-.444900000e-02  .622100000e-02  .317298000e+00  4.47963800  .200000000e-02

the A matrix is calculated from rhs as
 .000000e+00   .000000e+00   .000000e+00   .100000e+01   .000000e+00   .000000e+00   .000000e+00   .000000e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .100000e+01   .000000e+00   .000000e+00   .000000e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .100000e+01   .000000e+00   .000000e+00
-.947243e+00   .229538e+00  -.320625e+00   .182604e+03  -.142080e+02   .198669e+02  -.599265e+00  -.428304e-01
 .229538e+00  -.131666e-02  -.139499e+01  -.142080e+02   .124080e+03   .064007e+02  -.260619e+01  -.186260e+00
-.320625e+00  -.139499e+01   .948560e+00   .198669e+02   .864007e+02   .650572e+02   .364421e+01   .260457e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
now perturb each element of the state vector
 .000000e+00   .000000e+00   .000000e+00   .100000e+01   .000000e+00   .000000e+00   .000000e+00   .000000e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .100000e+01   .000000e+00   .000000e+00   .000000e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .100000e+01   .000000e+00   .000000e+00
-.947251e+00   .229545e+00  -.320661e+00   .182604e+03  -.142080e+02   .198679e+02  -.599265e+00  -.428303e-01
 .229527e+00  -.138355e-02  -.139515e+01  -.142073e+02   .124087e+03   .064048e+02  -.260619e+01  -.186260e+00
-.320610e+00  -.139504e+01   .948584e+00   .198660e+02   .064007e+02   .650635e+02   .364421e+01   .260457e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
 .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00

The Phi matrix check is as follows for
the initial state vector y =
-.132611000e+00 -.576970000e+00  .005928000e+00 -.102300000e-02
-.444900000e-02  .622100000e-02  .317298000e+00  4.47963800  .237183389e-02

phi is as follows after 50 iterations
   .999562     .000061    -.000085     .086631    -.005288     .007405    -.000107    -.000008
   .000061     .999814    -.000371    -.005289     .064859     .032195    -.000464    -.000034
  -.000005    -.000370    1.000066     .007401     .032167     .042799     .000649     .000048
  -.066115     .008841    -.011251   10.838960    -.764154    1.071041    -.011661    -.000873
   .008055    -.032944    -.049802    -.764620    7.692761    4.656234    -.050725    -.003796
  -.011222    -.040791     .000307    1.070107    4.649325    4.498297     .070888     .005305
   .000000     .000000     .000000     .000000     .000000     .000000    1.000000     .000000
   .000000     .000000     .000000     .000000     .000000     .000000     .000000    1.000000
calcu phi is as follows after 50 iterations
   .999562     .000061    -.000085     .086632    -.005287     .007405    -.000107    -.000008
   .000061     .999814    -.000371    -.005289     .064863     .032197    -.000464    -.000034
  -.000005    -.000370    1.000066     .007400     .032167     .042804     .000649     .000048
  -.066115     .008842    -.011254   10.839020    -.764117    1.071109    -.011661    -.000873
   .008055    -.032958    -.049016    -.764580    7.693633    4.656629    -.050725    -.003796
  -.011223    -.040799     .000302    1.069939    4.649112    4.499089     .070838     .005305
   .000000     .000000     .000003     .000000     .000000     .000000    1.000000     .000000
   .000000     .000000     .000000     .000000     .000000     .000000     .000000    1.000000

the H matrix check data is as follows for
the initial state vector y of
-.132611000e+00 -.576970000e+00  .005929000e+00 -.102300000e-02
-.444900000e-02  .622100000e-02  .317298000e+00  4.47963800  .200000000e-02

obser calculates H as follows
-.981747e+00   .177254e+00   .689589e-01   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
-.110144e+01  -.471653e+01  -.355784e+01   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
-.304365e+00  -.355600e+01   .400882e+01   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00

the perturbed calculation for H is as follows
-.981745e+00   .177421e+00   .687187e-01   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
-.110153e+01  -.471652e+01  -.355887e+01   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
-.304318e+00  -.35561e+01    .400894e+01   .000000e+00   .000000e+00   .000000e+00   .000000e+00   .000000e+00
```

V Results and Conclusions

ICBM Performance Parameters Estimation

With confidence that the programs are functioning correctly, as obtained from the results in the last chapter, the original problem of how to estimate the performance parameters of an ICBM in flight now begins. Due to time constraints caused by heavy computer usage, only one general case was examined, however the trends are quite certain to extend to the other cases which were programmed into the filter.

Sample test results are listed in Appendix J. The case which is presented uses 100 data point segments. Convergence is shown for all regions, and the covariance is listed. The case uses a radar site at 52.6° North, 174.1° East, 1 DU in elevation, and a launch point at 43° North, 132° East. A portion of the truth model observation data for the test case is listed in Table 15.

Table 15. Truth model data, ICBM test case

range (km)	azimuth (deg)	elevation (deg)	time (sec)
1065.0411	257.08497	-4.7879401	2.0136233
1065.0601	257.08497	-4.7866484	2.4131237
1065.0801	257.08497	-4.7853194	2.8136233
1065.0998	257.08495	-4.7839529	3.2136237
1065.1196	257.08494	-4.7825492	3.6136232
1065.1394	257.08494	-4.7811076	4.0136236

The truth model also numerically integrated the trajectory shown in Table 16. Notice that only the first 200

the simulated noisy data, with the covariance data for both trials listed in Table 13.

The tabulated data contains the corrections to the state from the last iteration of each Bayes loop. Again, all units are canonical unless specified. It is interesting to note that the filter converged on the perfect state within 2 iterations, and took only 1 additional iteration to converge on the other cases for the perfect data. Also the magnitude of the state corrections were almost all 10^{-15} which is very good. As with the least squares results, the data converged almost identically, with the only difference being the amount of time required to reach the solution, and the difference of order 10^{-8} is only slightly above the results from the least squares perfect data. The covariance matrices are again very small, with the error ellipsoid axis lengths decreasing as the filter processed more data. This is expected since the confidence in the estimate should go up as additional data is processed.

The simulated noisy calculations show the same trend as the noisy results for the least squares. The convergence took about one extra iteration, and the final errors were of order 10^{-3}, which is just above the least squares, noisy data results. The error ellipsoid axes again decreased with time, and were comparable with the perfect data results.

Table 14. Bayes Filter Test Results, Noisy Data

iteration	# 3	# 4	# 4
	loop 1	loop 2	loop 3
	-.1737332e-8	.3037524e-9	-.4274729e-9
	.5348930e-10	.1155548e-9	-.1121525e-9
CASE # 1	.7905271e-9	.4252964e-11	-.4957654e-10
	.2448065e-8	-.1910199e-8	.2261524e-9
	.7611670e-10	-.4510833e-9	-.1429128e-10
	-.1175545e-8	-.4293026e-10	-.3024271e-10
	# 3	# 4	# 4
	-.8691772e-9	.3037522e-9	-.4274728e-9
	-.3560272e-9	.1155548e-9	-.1121525e-9
	.3095512e-9	.4252958e-11	-.4957650e-10
Case # 2	.7122785e-8	-.1910192e-8	.2261527e-9
	.2991415e-9	-.4510835e-9	-.1429111e-10
	-.1698415e-8	-.4293023e-10	-.3024270e-10
	# 3	# 4	# 4
	.6331097e-9	.3037522e-9	-.4274728e-9
	-.1559395e-9	.1155548e-9	-.1121525e-9
CASE # 3	.5717885e-9	.4252939e-11	-.4957651e-10
	-.8614662e-8	-.1910198e-8	.2261527e-9
	.8116607e-9	-.4510833e-9	-.1429150e-10
	.4998575e-9	-.4293025e-10	-.3024260e-10
	# 3	# 4	# 4
	.8821206e-7	.3037518e-9	-.4274729e-9
	-.1524874e-7	.1155548e-9	-.1121525e-9
CASE # 4	-.2799510e-7	.4252960e-11	-.4957651e-10
	-.8882739e-7	-.1910197e-8	.2261525e-9
	.1766154e-7	-.4510830e-9	-.1429134e-10
	.3044111e-7	-.4293025e-10	-.3024286e-10

the final corrected states are:

Time 1	Time 2	Time 3
.2499575e+1	.2424369e+1	.2194183e+1
-.2669495e-3	.4396153e+0	.8536396e+0
-.8122994e-4	.4388443e+0	.8521426e+0
.2021882e-2	-.1570918e+0	-.3146224e+0
.4481786e+0	.4336677e+0	.3864206e+0
.4470668e+0	.4334313e+0	.3892277e+0

Table 13. Covariance data for Bayes Filter Tests (continued)

Noisy Observation Data

Covariance Matrix at epoch is:

.5180505e-13	-.1095052e-13	-.1781172e-13	-.1731339e-13	.1085341e-13	.1243912e-13
-.1095052e-13	.2469536e-14	.3835462e-14	.2185808e-14	-.2429973e-14	-.2439921e-14
-.1781172e-13	.3835462e-14	.6281015e-14	.4552161e-14	-.3752743e-14	-.4224094e-14
-.1731339e-13	.2185808e-14	.4552161e-14	.3859435e-13	-.3387219e-14	-.8616803e-14
.1085341e-13	-.2429973e-14	-.3752743e-14	-.3387219e-14	.2500323e-14	.2593289e-14
.1243912e-13	-.2439921e-14	-.4224094e-14	-.8616803e-14	.2593289e-14	.3841942e-14
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00

the error ellipsoid axis lengths for the covariance matrix are
.137829e+01 meters
.105405e-02 meters
.000000e+00 meters
the error ellipsoid axis lengths for the position components are
.156509e+01 meters
.893599e-01 meters
.536925e-01 meters

Covariance Matrix at epoch is:

.5689967e-13	.2893223e-15	-.7439639e-14	-.1623987e-13	.9627018e-14	.1052807e-13
.2893223e-15	.1110523e-15	.1536843e-16	-.1103610e-14	-.2911966e-15	-.5215173e-16
-.7439639e-14	.1536843e-16	.1080156e-14	.1241479e-14	-.1496019e-14	-.1554642e-14
-.1623987e-13	-.1103610e-14	.1241479e-14	.2469651e-13	.2350799e-14	-.1188691e-14
.9627018e-14	-.2911966e-15	-.1496019e-14	.2350799e-14	.3067733e-14	.2271053e-14
.1052807e-13	-.5215173e-16	-.1554642e-14	-.1188691e-14	.2271053e-14	.2293103e-14
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00

the error ellipsoid axis lengths for the covariance matrix are
.147560e+01 meters
.421826e-03 meters
.421826e-03 meters
.627327e-03 meters/sec
the error ellipsoid axis lengths for the position components are
.153441e+01 meters
.472052e-01 meters
.808000e-01 meters

Covariance Matrix at epoch is:

.5294432e-13	.1316609e-13	.5108883e-14	-.1688682e-13	.8289553e-14	.8774566e-14
.1316609e-13	.3348274e-14	.1315473e-14	-.4725914e-14	.1694544e-14	.1954117e-14
.5108888e-14	.1315473e-14	.5723077e-15	-.2019687e-14	.5399098e-15	.6101817e-15
-.1688682e-13	-.4725914e-14	-.2019687e-14	.1213249e-13	.1478759e-14	-.1545602e-15
.8289553e-14	.1694544e-14	.5399098e-15	.1478759e-14	.3894437e-14	.3032544e-14
.8774566e-14	.1954117e-14	.6101817e-15	-.1545602e-15	.3032544e-14	.2652090e-14
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00

the error ellipsoid axis lengths for the covariance matrix are
.142217e+01 meters
.544872e-00 meters
.389460e-03 meters
.482731e-03 meters/sec
the error ellipsoid axis lengths for the position components are
.151900e+01 meters
.684679e-01 meters

Table 13. Covariance Data for Bayes Filter Tests

Perfect Observation Data

Covariance Matrix at epoch is:

.5185616e-13	-.1095270e-13	-.1782283e-13	-.1738451e-13	.1084102e-13	.1245239e-13
-.1095270e-13	.2467963e-14	.3834787e-14	.2199219e-14	-.2424886e-14	-.2440795e-14
-.1782283e-13	.3834787e-14	.6282652e-14	.4572748e-14	-.3746558e-14	-.4226799e-14
-.1738451e-13	.2199219e-14	.4572748e-14	.3867913e-13	-.3407667e-14	-.3640367e-14
.1084102e-13	-.2424886e-14	-.3746558e-14	-.3407667e-14	.2492565e-14	.2592311e-14
.1245239e-13	-.2440795e-14	-.4226799e-14	-.3640367e-14	.2592311e-14	.3846068e-14
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00

the error ellipsoid axis lengths for the covariance matrix are
.137874e+01 meters
.105428e-02 meters
.000000e+00 meters
the error ellipsoid axis lengths for the position components are
.156656e+01 meters
.893346e-01 meters
.586845e-01 meters

Covariance Matrix at epoch is:

.5686771e-13	.2867172e-15	-.7425419e-14	-.1618743e-13	.9636801e-14	.1052816e-13
.2867172e-15	.1106718e-15	.1562240e-16	-.1098916e-14	-.2904672e-15	-.5265900e-16
-.7425419e-14	.1562240e-16	.1076660e-14	.1234741e-14	-.1494939e-14	-.1552548e-14
-.1618743e-13	-.1098916e-14	.1234741e-14	.2458098e-13	.2335858e-14	-.1180364e-14
.9636801e-14	-.2904672e-15	-.1494939e-14	.2335858e-14	.3067210e-14	.2274167e-14
.1052816e-13	-.5265900e-16	-.1552548e-14	-.1180364e-14	.2274167e-14	.2294373e-14
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00

the error ellipsoid axis lengths for the covariance matrix are
.147528e+01 meters
.421684e-03 meters
.421684e-03 meters
.625286e-03 meters/sec
the error ellipsoid axis lengths for the position components are
.153395e+01 meters
.471192e-01 meters
.806310e-01 meters

Covariance Matrix at epoch is:

.5262459e-13	.1305212e-13	.5072951e-14	-.1654805e-13	.8405727e-14	.8808699e-14
.1305212e-13	.3311482e-14	.1303232e-14	-.4633724e-14	.1716350e-14	.1955901e-14
.5072951e-14	.1303232e-14	.5682836e-15	-.1986832e-14	.5495807e-15	.6121819e-15
-.1654805e-13	-.4633724e-14	-.1986832e-14	.1200593e-13	.1510901e-14	-.1065965e-15
.8405727e-14	.1716350e-14	.5495807e-15	.1510901e-14	.3956834e-14	.3076475e-14
.8808699e-14	.1955901e-14	.6121819e-15	-.1065965e-15	.3076475e-14	.2679237e-14
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00
.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00	.0000000e+00

the error ellipsoid axis lengths for the covariance matrix are
.141731e+01 meters

.141731e+01 meters
.547602e-00 meters
.387014e-00 meters/sec
.479700e-03 meters/sec
the error ellipsoid axis lengths for the position components are
.151416e+01 meters
.684823e-01 meters
.356544e-01 meters

Table 12. Bayes Filter Test Results, Perfect Observations

Corrections to the state from the last iteration:			
iteration	loop 1	loop 2	loop 3
	# 2	# 2	# 2
CASE # 1	-.3524090e-15	.3775959e-15	.7204185e-16
	.1607043e-15	-.3591601e-16	.1625420e-16
	.7630261e-16	.4048324e-17	.8456722e-17
	.7847741e-15	-.1044663e-14	-.6064243e-15
	-.1693562e-15	.6318910e-16	-.4719534e-15
	-.1939084e-15	.3291155e-18	-.1421949e-15
	# 3	# 2	# 2
CASE # 2	-.6998720e-13	.5343816e-15	.7503202e-16
	.1195828e-14	.1436418e-15	.7101780e-16
	.1974012e-14	-.6164276e-16	.4390330e-16
	.9340976e-13	-.1522799e-14	-.4117666e-15
	.4582638e-14	-.2906382e-15	-.6517259e-15
	.4104247e-14	.6863711e-16	-.1882977e-15
	# 3	# 2	# 2
CASE # 3	.2342559e-14	.4178976e-15	.2650948e-15
	-.5523967e-15	.6595235e-16	.1126490e-15
	-.1012912e-14	-.6635110e-16	.3337257e-16
	-.3881522e-14	-.1287997e-14	-.8903416e-15
	.1229017e-14	-.1083272e-15	-.7648698e-15
	.2243644e-14	.7944296e-16	-.1241624e-15
	# 3	# 2	# 2
CASE # 4	-.1011434e-8	.1709862e-15	.3549278e-16
	.5786753e-9	-.9786427e-19	-.5614404e-17
	.7691572e-9	-.5444362e-17	.1247841e-16
	.1788590e-8	-.3533691e-15	-.1854145e-15
	-.8509662e-10	.5039547e-16	-.3902865e-15
	-.7039212e-9	.4078983e-16	-.6353586e-16

The final corrected states are:		
Time 1	Time 2	Time 3
.2500000e+1	.2421457e+1	.2190766e+1
.2511528e-8	.4396257e+0	.8516282e+0
-.1926930e-8	.4396257e+0	.8516282e+0
-.7865713e-8	-.1572853e+0	-.3046877e+0
.4472135e+0	.4331635e+0	.3918962e+0
.4472135e+0	.4331635e+0	.3918962e+0

Table 11. Least Squares Test Results, Noisy Observations

CASE # 1 Corrections for iteration #

1	2	3
.7924433e-4	-.4206869e-8	.1216706e-11
.1077903e-4	-.3045991e-8	-.1852942e-12
.9042774e-5	-.7908962e-9	.2482118e-11
-.1686651e-4	.1018113e-8	-.4850761e-12
-.1781953e-4	.1238650e-8	-.9031411e-13
-.3911945e-5	.3462006e-9	-.3183258e-12

CASE # 2 Corrections for iteration #

1	2	3	4
.1538294e-2	.4096174e-4	-.1647061e-7	-.1540987e-10
.2617755e-5	.8171601e-5	-.1336570e-7	.2158184e-11
-.1334271e-3	.1424602e-3	.8914906e-8	-.3576532e-10
-.2302934e-5	-.1456163e-4	-.9364774e-9	.6372505e-11
.3163803e-4	-.4945985e-4	.3522301e-8	.5392571e-12
-.3960326e-4	.3569532e-4	-.3660874e-8	.4893863e-11

CASE # 3 Corrections for iteration #

1	2	3	4
.6927874e-4	.9976133e-5	-.1474618e-7	-.1859578e-11
.1848499e-4	-.7699773e-5	-.9227279e-8	-.2557389e-11
-.4008284e-4	.4913705e-4	-.1221827e-7	-.1033730e-11
-.1013049e-2	-.3820235e-5	.3970187e-8	.5265535e-12
-.3238347e-5	-.1458009e-4	.1520327e-9	-.7616239e-13
-.1385391e-4	.9938250e-5	.4062703e-8	.6695699e-12

CASE # 4 Corrections for iteration #

1	2	3	4
.1465780e-2	.1135676e-3	-.1081471e-6	-.1593583e-10
-.1008484e-1	.9563121e-4	-.1354342e-7	-.6378967e-11
-.1009848e-1	.1076447e-3	-.1127688e-6	-.2897397e-10
-.9720443e-3	-.4485862e-4	.3746891e-7	.6037466e-11
.5100931e-3	-.8061368e-4	-.2256346e-8	-.4146842e-12
.4021184e-3	.4123910e-4	.2633033e-7	.5925330e-11

The final corrected state was:

.2500079e+1
.1077599e-4
.9041986e-5
-.1686550e-4
.4471957e+0
.44720968+0

the actual solution. Finally, the relative magnitude of the differences between the exact and the estimated state are all about order 10^{-9} which are the same order as seen in the numerical integration checkout.

To complete checkout of the least squares filter, the same four cases were tested with the simulated noisy data from the truth model. Table 11 lists the results. In general, the results were similar to those with the perfect data: however, it generally took longer for the filter to eliminate the noise which was present in the data, and the final state had errors of order 10^{-4}. This represents a significant difference to the least squares results. As can be seen in Table 10, the covariance matrix was also the same for all the cases, and almost identical to the perfect data case.

Bayes Filter Checkout

The last check which was performed was to check the Bayes filter estimator. Again, for consistency, the same truth model data was used. Here however, there was an effort to limit the computational time due to extremely heavy usage of the computer, therfore, only 3 iterations of the Bayes filter were tested, with each of the runs processing 30 data points. This would require only a few matrix inversions, and proved adequate to show trends in the performance of the filter. The four cases were tested against the perfect data, and then the simulated noisy data. Table 12 lists the perfect data tests and Table 14 lists the Bayes filter estimations of

Table 10. Covariance Data for Least Squares Tests

```
PERFECT OBSERVATION DATA

 Covariance Matrix at epoch is:
  .1894375e-17  .1906222e-17  .1949674e-17 -.7362794e-18 -.2880636e-18 -.2487960e-18
  .1906222e-17  .3105600e-17  .1954685e-17 -.8456409e-18 -.3394847e-18 -.2068683e-18
  .1949674e-17  .1954685e-17  .3374542e-17 -.9002667e-18 -.2249767e-18 -.3332718e-18
 -.7362794e-18 -.8456409e-18 -.9002667e-18  .3262177e-18  .1080523e-18  .9784323e-19
 -.2880636e-18 -.3394847e-18 -.2249767e-18  .1080523e-18  .6719715e-19  .1457902e-19
 -.2487960e-18 -.2068688e-18 -.3332718e-18  .9784323e-19  .1457902e-19  .5707239e-19
  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00
  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00

 the error ellipsoid axis lengths for the covariance matrix are
  .772805e-02 meters
  .565357e-05 meters
  .565357e-05 meters
 the error ellipsoid axis lengths for the position components are
  .165522e-01 meters
  .380495e-02 meters
  .722606e-02 meters
```

```
NOISY OBSERVATION DATA

 Covariance Matrix at epoch is:
  .1894579e-17  .1906361e-17  .1949884e-17 -.7363331e-18 -.2880884e-18 -.2488251e-18
  .1906361e-17  .3105715e-17  .1954824e-17 -.8456602e-18 -.3395106e-18 -.2060873e-18
  .1949884e-17  .1554824e-17  .3374814e-17 -.9003164e-18 -.2250010e-18 -.3333097e-18
 -.7363331e-18 -.8456602e-18 -.9003164e-18  .3262245e-18  .1080591e-18  .9785187e-19
 -.2880884e-18 -.3395106e-18 -.2250010e-18  .1080591e-18  .6719833e-19  .1458403e-19
 -.2488251e-18 -.2068873e-18 -.3333097e-18  .9785187e-19  .1458483e-19  .5707367e-19
  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00
  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00  .0000000e+00

 the error ellipsoid axis lengths for the covariance matrix are
  .772809e-02 meters
  .565409e-05 meters
  .565409e-05 meters
 the error ellipsoid axis lengths for the position components are
  .165529e-01 meters
  .380508e-02 meters
  .722700e-02 meters
```

Table9.LeastSquares Test Results PerfectObservations

CASE # 1 Corrections for iteration #			
	1	2	
	.2121908e-8	.6499638e-15	
	-.5513195e-9	.1664450e-15	
	-.7417027e-9	.1357164e-15	
	-.1253529e-9	-.9853450e-16	
	-.2348516e-9	-.6637222e-16	

CASE # 2 Corrections for iteration #			
1	2	3	4
.1461205e-2	.3878021e-4	.1600352e-7	.1350803e-13
-.8012713e-5	.7986139e-5	.2602269e-7	.2013430e-14
-.1387883e-3	.1387741e-3	.1345843e-7	.3570665e-13
.1370964e-4	-.1370099e-4	-.8527314e-8	-.5377310e-14
.4933845e-4	-.4934259e-4	.4015420e-8	-.7667879e-14
-.3623750e-4	.3625131e-4	-.1405055e-7	.2879627e-14

CASE # 3 Corrections for iteration #			
1	2	3	4
-.1106544e-4	.1106884e-4	-.1277582e-8	-.6457492e-15
.7545843e-5	-.7549170e-5	.2775568e-8	-.2021483e-15
-.5094762e-4	.5095004e-4	-.3154845e-8	-.4438048e-15
-.9957467e-3	-.4253397e-5	.2297846e-9	.2871613e-15
.1462024e-4	-.1462056e-4	.1873276e-9	-.5522853e-16
-.9643821e-5	.9643834e-5	-.2476946e-9	.2050013e-15

CASE # 4 Corrections for iteration #			
1	2	3	4
.1388528e-2	.1115167e-3	-.4337689e-7	.9875105e-14
-.1009635e-1	.9632617e-4	.3166193e-7	.5226998e-14
-.1010382e-1	.1038821e-3	-.5419912e-7	.2427022e-13
-.9561011e-3	-.4391646e-4	.1776285e-7	-.3653104e-14
.5278604e-3	-.8056149e-4	-.3599163e-8	-.9022306e-14
.4055274e-3	.4176069e-4	.7124101e-8	.5774726e-14

The final corrected state was equal to:
.2500000e+1
-.5512194e-9
-.7417025e-9
.1283654e-9
.4472135e+0
.4472135e+0

Each of the cases was run with the least squares filter and the results are shown in Table 9. (units are canonical and only 6 states are shown since the satellite data consisted only of position and velocity vectors.) Note that the filter converged in only 2 iterations on the perfect data case, yet it required 4 iterations on the perturbed initial state vectors. In each case, the final state was the same since the estimator was using the same truth model data. The filter, after processing all 500 data points of one period of the orbit, improves the initial guess by a very small amount, 10^{-9}, which can really be treated as truncation and machine error. A measure of the accuracy of the filter is the covariance matrix, from which the error ellipsoid axis lengths can be determined. The procedure here is to calculate the eigenvalues of the covariance matrix. The square root of each eigenvalue is then the error ellipsoid axis length. Table 10 lists the covariance matrix and the error ellipsoid axis lengths. The covariance was the same for all the different cases using the same truth model data, which simply means that the same degree of confidence can be placed with each estimate. This of course assumes that the dynamics model is valid. The second set of axis lengths are considered to be more accurate since they represent the upper left 3x3 partition of the covariance matrix. This is the best measure of the accuracy of the position components. The filter seemed to adapt rapidly to the errors, correct them, and converge on

The truth model program generated the numerical integration data shown in Table 3 and also produced the truth model (range, azimuth and elvation) data for the orbit. Table 7 lists the first few lines, and the last few lines of the truth model data as it was used. (both perfect and noisy data simulations are included)

Table 7. Truth model data for satellite orbit

simulated perfect observation data			
range (DU)	azimuth (rad)	elevation (rad)	time (TU)
2.10828311	5.43568181	.190435864e+0	.496729413e-1
2.09648962	5.43864117	.203551598e+C	.993458827e-1
2.08473224	5.44165050	.216743101e+0	.149018824e+0
2.07301339	5.44471208	.230011201e+0	.198691765e+0
...			
2.29091161	5.21078168	-.714578847e-3	.246874518e+2
2.27868763	5.21196854	.115080426e-1	.247371248e+2
simulated noisy observation data:			
2.10828823	5.43455972	.189748349e+0	.496729413e-1
2.09648944	5.43893145	.204563277e+0	.993458827e-1
2.08473754	5.44310397	.216215598e+0	.149018824e+0
2.07303439	5.44533297	.231198678e+0	.198691765e+0
...			
2.29090397	5.21055858	.940219542e-3	.246874518e+2
2.27869430	5.21154640	.127423844e-1	.247371248e+2

For each test, four cases of initial input data vectors were run. Table 8 summarizes the test cases.

Table 8. Filter test cases

case 1	case 2	case 3	case 4
.2500000e+1	.2498000e+1	.2500000e+1	.2498[illegible]0e+1
.0000000e+0	.0000000e+0	.0000000e+0	.1000000e-2
.0000000e+0	.0000000e+0	.0000000e+0	.1000000e-2
.0000000e+0	.0000000e+0	.1000000e-2	.1000000e-2
.4472135e+0	.4472135e+0	.4472135e+0	.4467663e+0
.4472135e+0	.4472135e+0	.4472135e+0	.4467663e+0

It is important to realize that all the calculations must be made in the IJK frame. There are several places in the programs where conversions between SEZ and IJK frames are performed, and later when runs are made with the Least Squares and the Bayes programs, their input data, in order to be consistent, must be all in the IJK frame.

Least Squares Checkout

The next step was to run a trial program of Least Squares. This would serve only to check the combination of all the different matrices and formulations that were developed thus far. The program estimates the original state vector, given only an initial guess, and the range, azimuth, elvation, and time data from the truth model. The program is listed in Appendix H. For consistency, all of the test cases for the Least Squares and the Bayes filter were identical, using the following input data.

Table 6. Test Cases, Input Data

Satellite orbit $a = 2.5$ DU $e = 0$ $i = 45$ deg $\Omega = 0$ deg $\omega = 0$ deg $M = 0$ deg
radar site 45 deg North 60 deg West 1.0225DU elevation
dt = period/500 TU $t_o = 0$ TU initial velocity 200 ft/sec displacement of .06 DU/TU

in the calculated matrix was also about 1×10^{-4}.

Next, the ϕ matrix was calculated and checked . See Appendix E for the checkout program. This was accomplished by first setting the state vector to an initial value and saving it. The state was then moved through time (which worked out to be about $^1/_4$ of the orbit), and again saved. Individually then, each of the elements of the state vector were perturbed at the original time and translated through time. The columns of the ϕ matrix could then be calculated as:

$$\phi = \frac{X(\bar{x}(t_o) + \delta,\ t) - X(\bar{x}(t_o), t)}{\delta} \tag{4-2}$$

It should be noted here that after each perturbation, the state and the ϕ matrix were reinitialized. Table 5 lists the output and shows that, for input errors to the state vector of about 1×10^{-4}, the same order of magnitude errors were observed in the calculated ϕ matrix.

Once the G and H matrices were programmed, a check was performed on the H matrix. The program is listed in Appendix F. The state vector is input, and one call to obser calculates the H matrix directly. Then, similar to the check for the A matrix, each one of the state vector elements are perturbed, and the columns of the H matrix are calculated as:

$$[\ H\ ] = \frac{G(\bar{x} + \delta,\ t\ ) - G(\bar{x}\ ,\ t)}{\delta} \tag{4-3}$$

Table 16. Numerical Integration of ICBM Test Case

The initial state vector for the missile is

x	y	z	xdot
-.1326113091294e+00	-.5769695352121e+00	.8059282822486e-00	-.1022594408953e-02

ydot	zdot	Ve	M
-.4449136538335e-02	.6276833691360e-02	.3172982551730e+00	.4479637558632e+01

the initial time is .0020 TU

r (km)	v (ft/sec)	gamma (deg)	time (sec)	ve (DU/TU)	m
6378.425311	259.243619	.4765268340	5.6136237488	.3172982552	4.4796375586
6378.778878	321.528788	.7435895547	9.6136237488	.3172982552	4.4796375586
6379.211102	388.369675	1.0696148558	13.6136237488	.3172982552	4.4796375586
6379.727641	459.996128	1.4495086812	17.6136237488	.3172982552	4.4796375586
6380.334418	536.656672	1.8770713579	21.6136237488	.3172982552	4.4796375586
6381.037640	618.620109	2.3475806240	25.6136237488	.3172982552	4.4796375586
6381.843819	706.177191	2.8481994209	29.6136237488	.3172982552	4.4796375586
6382.759789	799.642473	3.3782525931	33.6136237488	.3172982552	4.4796375586
6383.792733	899.356393	3.9294071189	37.6136237488	.3172982552	4.4796375586
6384.950208	1005.687698	4.4957822772	41.6136237488	.3172982552	4.4796375586
6386.240181	1119.036281	5.0720095664	45.6136237488	.3172982552	4.4796375586
6387.671061	1239.836538	5.6532571069	49.6136237488	.3172982552	4.4796375586
6389.251743	1368.561347	6.2352294052	53.6136237488	.3172982552	4.4796375586
6390.991655	1505.726783	6.8141504768	57.6136237488	.3172982552	4.4796375586
6392.900812	1651.897727	7.3867361718	61.6136237488	.3172982552	4.4796375586
6394.989882	1807.694513	7.9501599517	65.6136237488	.3172982552	4.4796375586
6397.270250	1973.800849	8.5020151758	69.6136237488	.3172982552	4.4796375586
6399.754111	2150.973260	9.0402760696	73.6136237488	.3172982552	4.4796375586
6402.454552	2340.052397	9.5632588936	77.6136237488	.3172982552	4.4796375586
6405.356695	2490.305197	10.0747264442	81.6136237488	.3929044800	3.5272654570
6408.436116	2645.832961	10.5795952776	85.6136237488	.3929044800	3.5272654570
6411.701670	2809.651417	11.0765767476	89.6136237488	.3929044800	3.5272654570
6415.162763	2982.238499	11.5645042960	93.6136237488	.3929044800	3.5272654570
6418.829345	3164.112753	12.0423381122	97.6136237488	.3929044800	3.5272654570
6422.711963	3355.839115	12.5091584371	101.6136237488	.3929044800	3.5272654570
6426.821817	3558.034961	12.9641583753	105.6136237488	.3929044800	3.5272654570
6431.170828	3771.377151	13.4066362980	109.6136237488	.3929044800	3.5272654570
6435.771714	3996.610301	13.8359880170	113.6136237488	.3929044800	3.5272654570
6440.638078	4234.556504	14.2516988823	117.6136237488	.3929044800	3.5272654570
6445.784503	4486.126839	14.6533359156	121.6136237488	.3929044800	3.5272654570
6451.226663	4752.335075	15.0405400725	125.6136237488	.3929044800	3.5272654570
6456.981461	5034.314065	15.4130186941	129.6136237488	.3929044800	3.5272654570
6463.067170	5333.335523	15.7705381906	133.6136237488	.3929044800	3.5272654570
6469.503616	5650.834031	16.1129169784	137.6136237488	.3929044800	3.5272654570
6476.312379	5988.436427	16.4400186731	141.6136237488	.3929044800	3.5272654570
6483.517043	6347.998092	16.7517455262	145.6136237488	.3929044800	3.5272654570
6491.143480	6731.648165	17.0480320723	149.6136237488	.3929044800	3.5272654570
6499.220206	7141.846486	17.3288389399	153.6136237488	.3929044800	3.5272654570
6507.778798	7581.456101	17.5941467558	157.6136237488	.3929044800	3.5272654570
6516.854418	8053.836767	17.8439500500	161.6136237488	.3929044800	3.5272654570
6526.486452	8562.967167	18.0782510354	165.6136237488	.3929044800	3.5272654570
6536.719313	9113.607185	18.2970530935	169.6136237488	.3929044800	3.5272654570
6547.603469	9711.517058	18.5003537347	173.6136237488	.3929044800	3.5272654570
6559.196754	10363.759207	18.6881367086	177.6136237488	.3929044800	3.5272654570
6571.566111	11079.123248	18.8603627994	181.6136237488	.3929044800	3.5272654570
6584.789908	11868.739931	19.0169586114	185.6136237488	.3929044800	3.5272654570
6598.961134	12746.994767	19.1578022810	189.6136237488	.3929044800	3.5272654570
6614.191898	13732.936069	19.2827044076	193.6136237488	.3929044800	3.5272654570
6630.620006	14852.537972	19.3913813462	197.6136237488	.3929044800	3.5272654570
6648.418941	16142.528699	19.4834158043	201.6136237488	.3929044800	3.5272654570

seconds are shown since it was desired to limit the computational time. The staging event between stage I and stage II could then be observed.

A main concern was to show that the filter could not only observe the first stage parameters, since it would surely work on these, given the results of the last chapter, but that it could also observe the staging event, and continue to estimate the second stage parameters as additional data was processed. If the filter could observe the staging event of the first and second stages, it would also be able to observe the staging events throughout the rest of the flight.

Several test cases, such as those listed in Appendix J were run. It was desired to have the filter retain good memory about the position and velocity, since the changes in these quantities, even during staging events, would be small. However, the V_e and M would be changing rapidly only at the staging events. To account for this phenomena, the test cases were run with β values of .8 - 1.0 for the position and velocity vectors, and .5 - .7 for the V_e and M. The effect of this was to have the filter try to throttle the ICBM in order to make the estimation correct when it was close to a staging event. The compiled data is graphed in Figures 6 thru 14, and shown numerically in Appendix J. Notice that in the figures, the best results are shown in Figures 6 and 10 (exhaust velocity and mass ratio respectively), and the

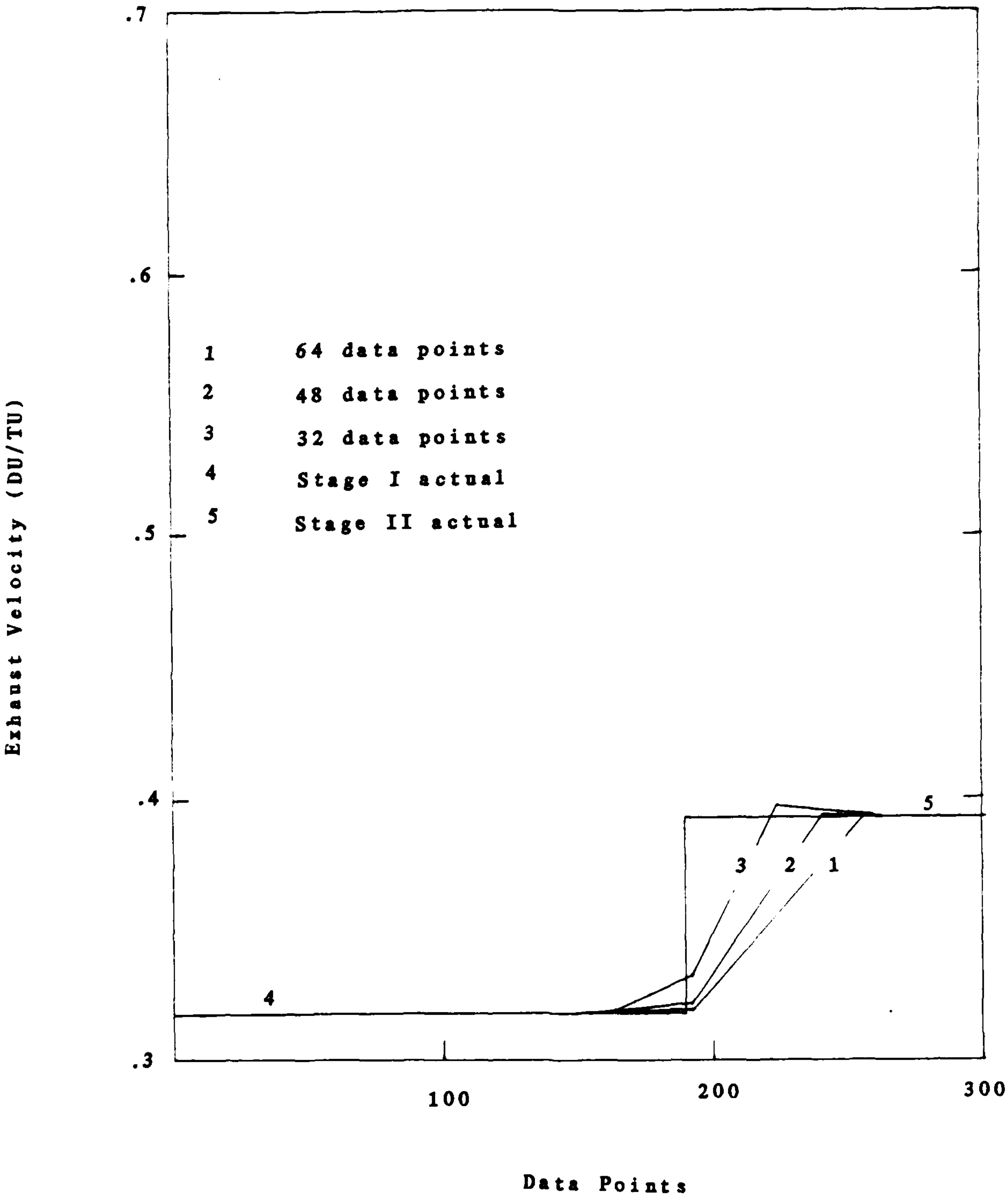


Figure 6. Test Cases: 64, 48, 32

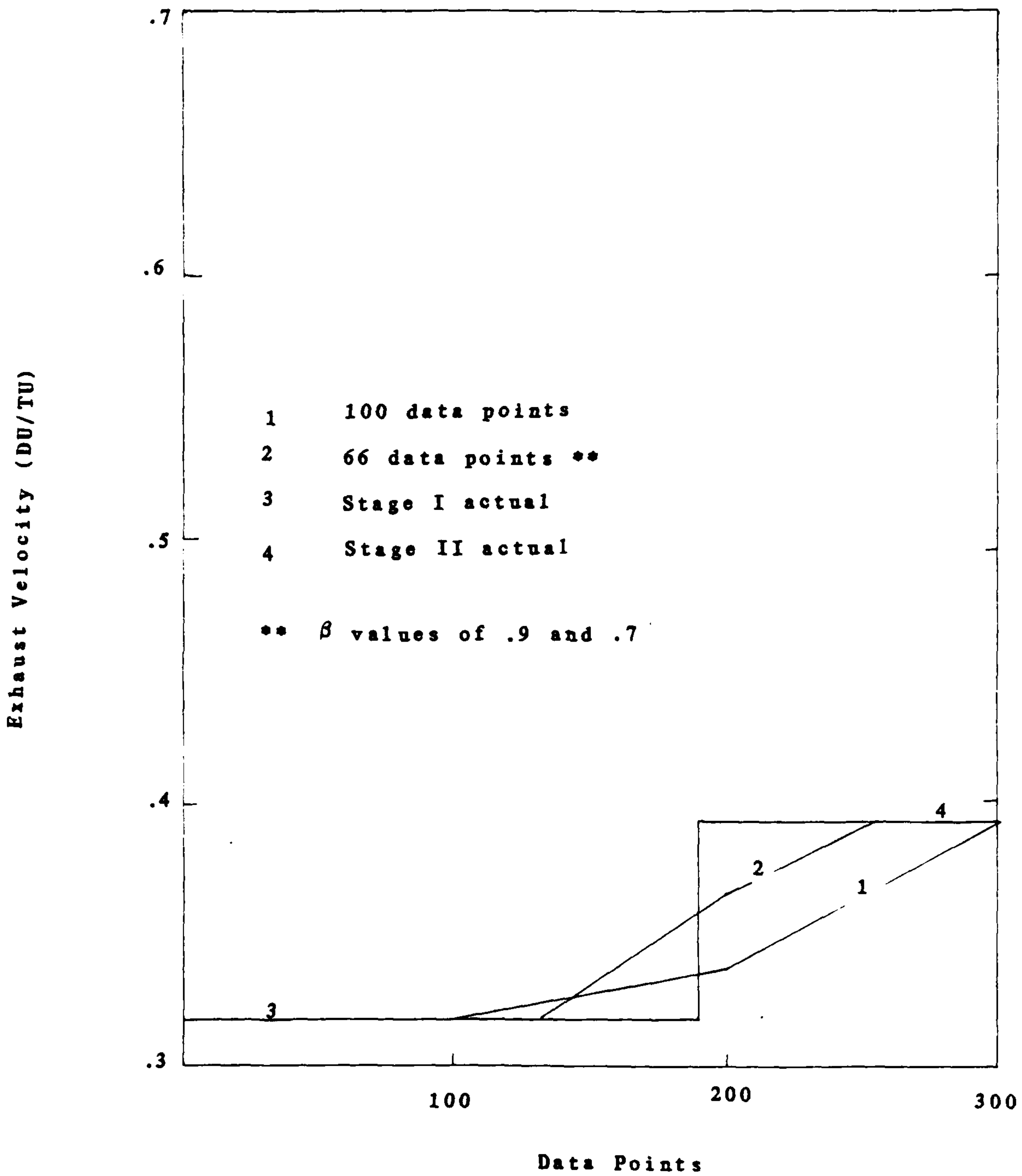


Figure 7. Test Cases: 100, 66

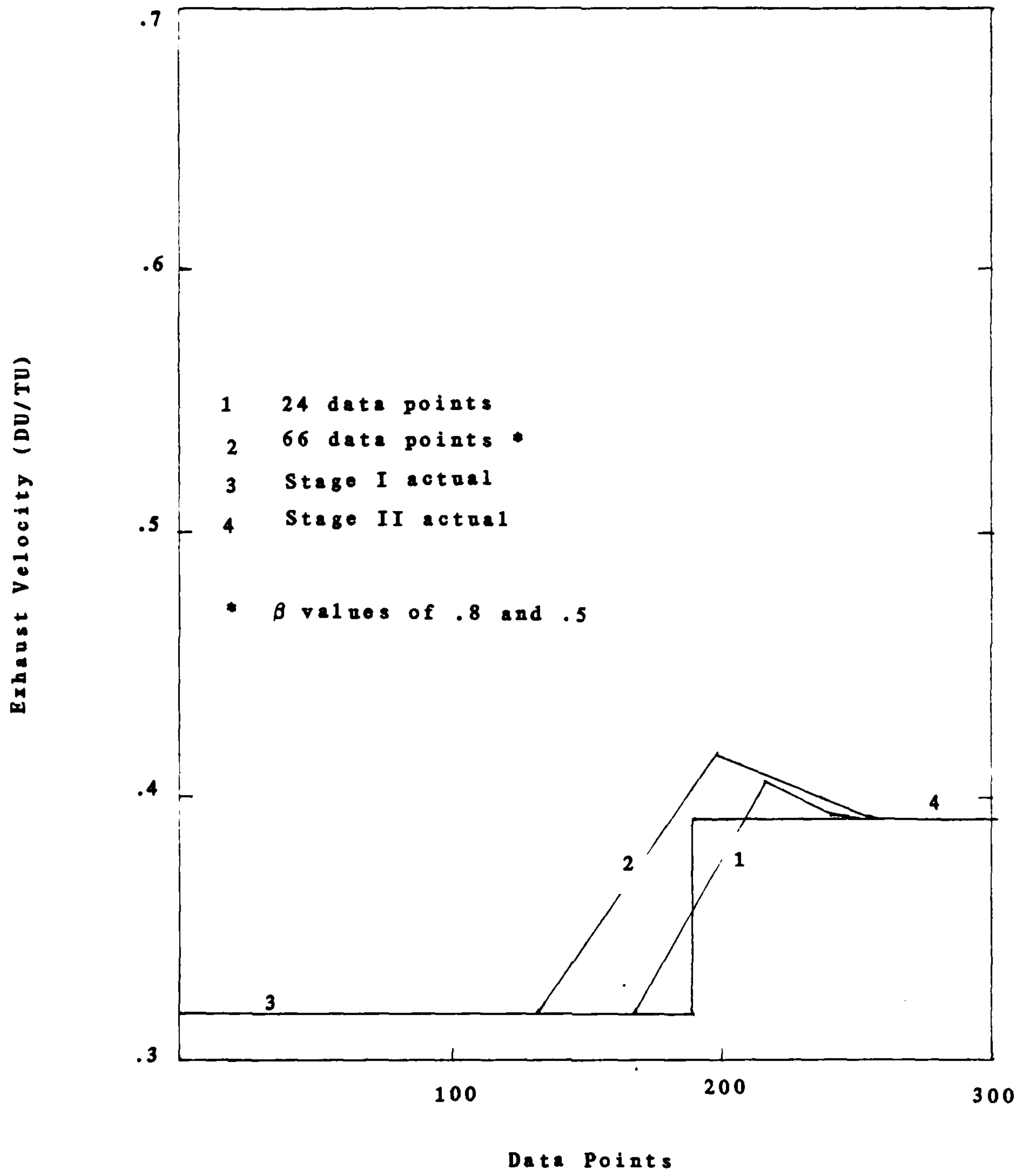


Figure 8. Test Cases: 24, 66

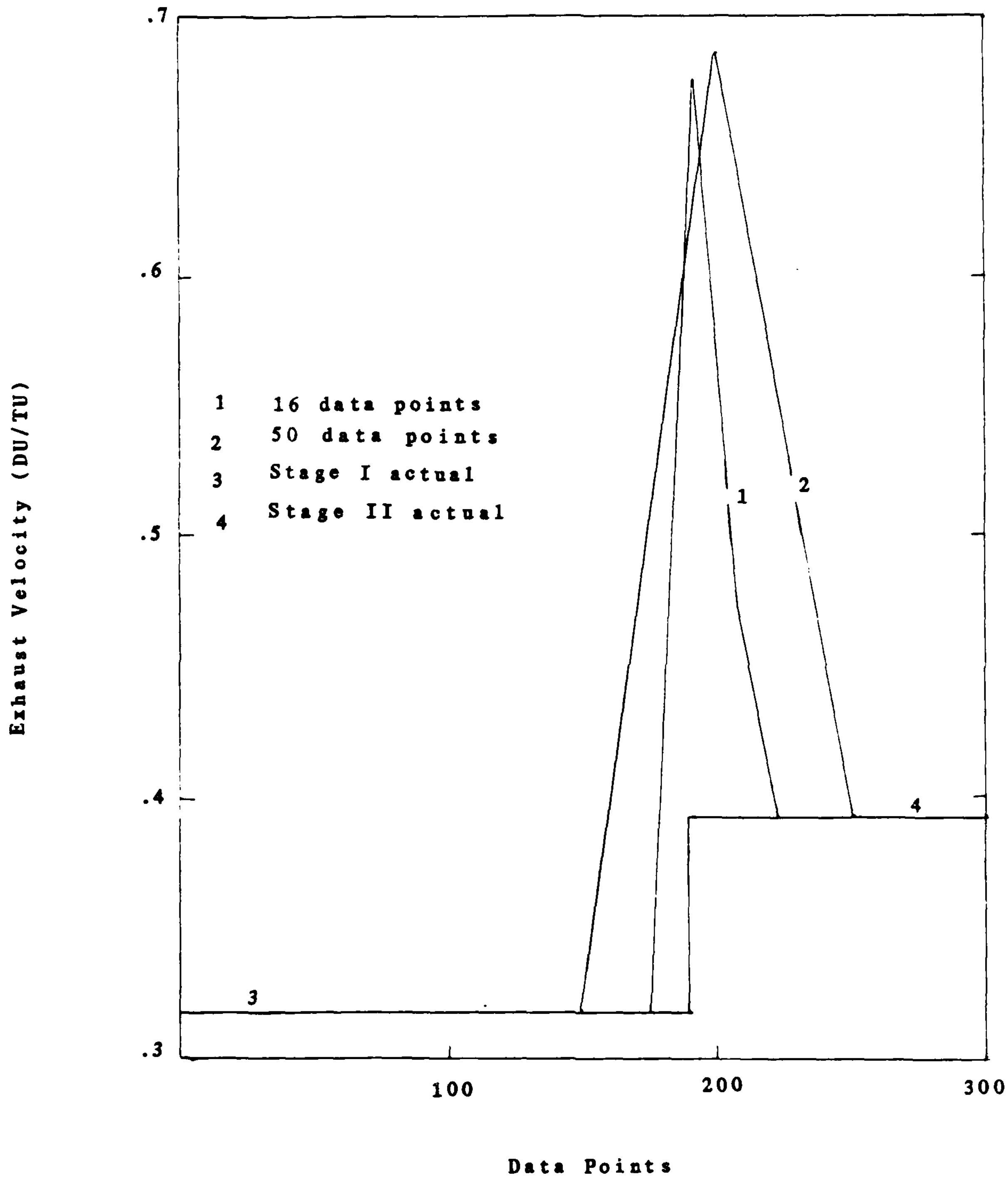


Figure 9. Test Cases: 16, 50

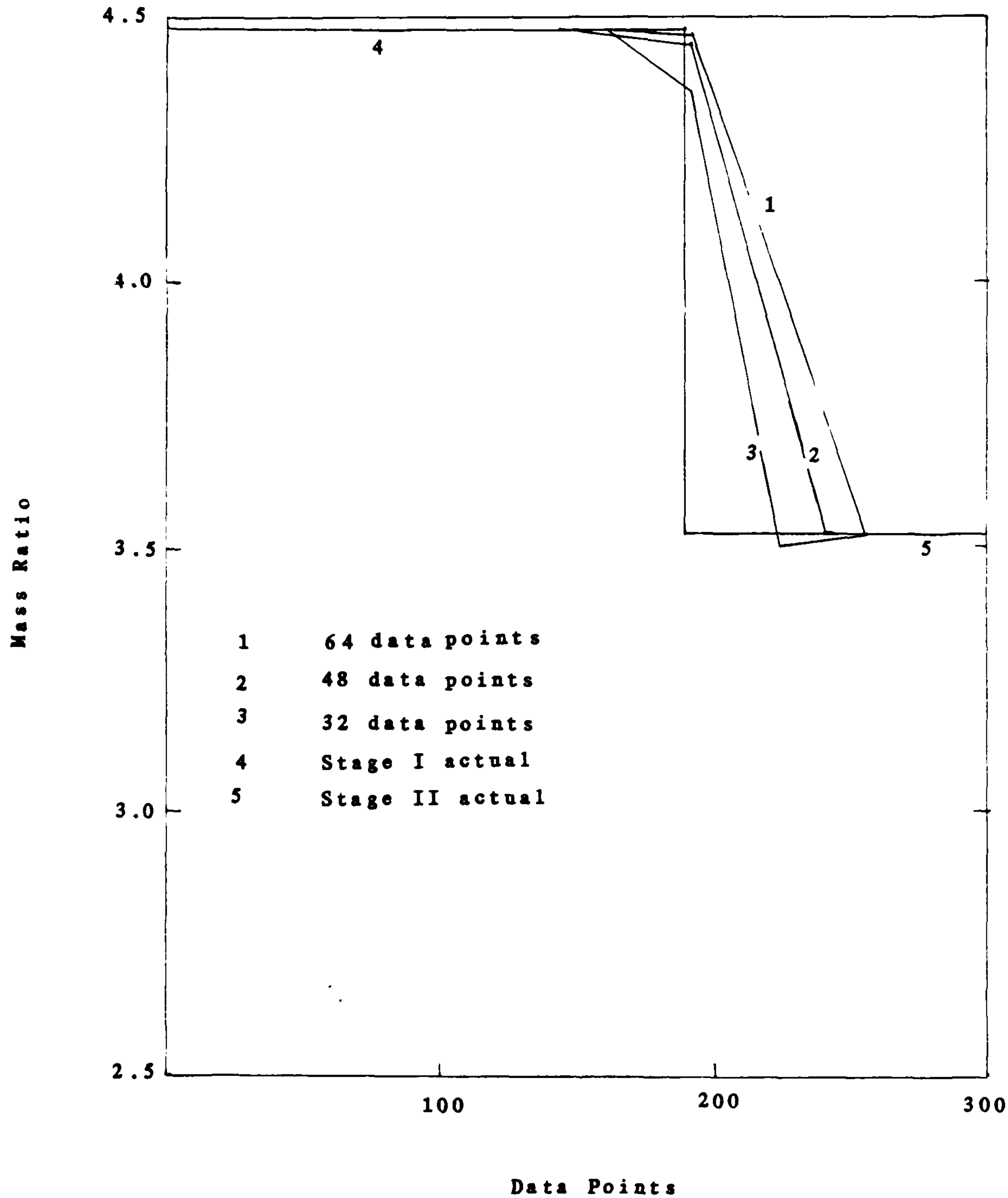


Figure 10. Test Cases: 64, 48, 32

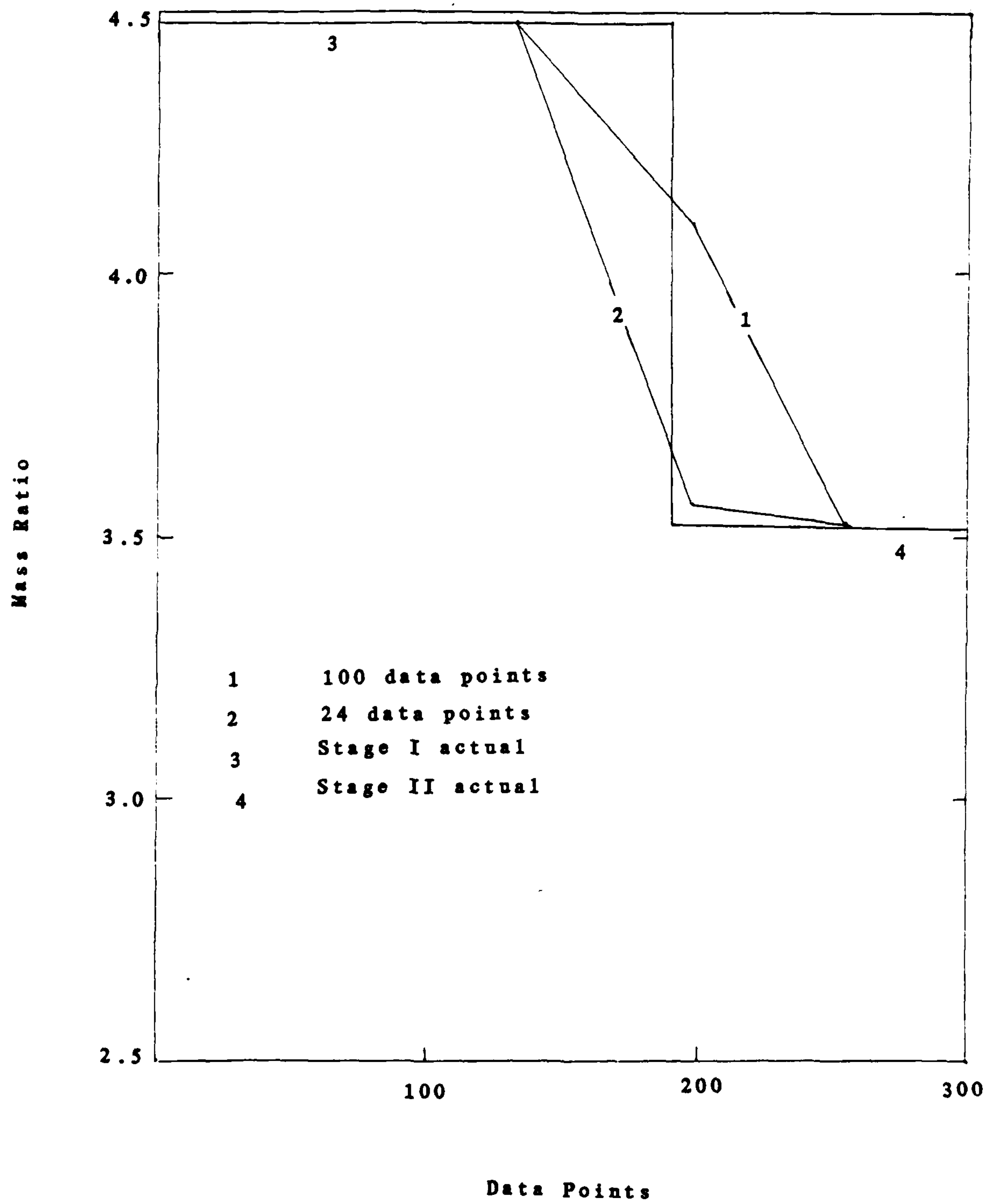


Figure 11. Test Cases: 100, 24

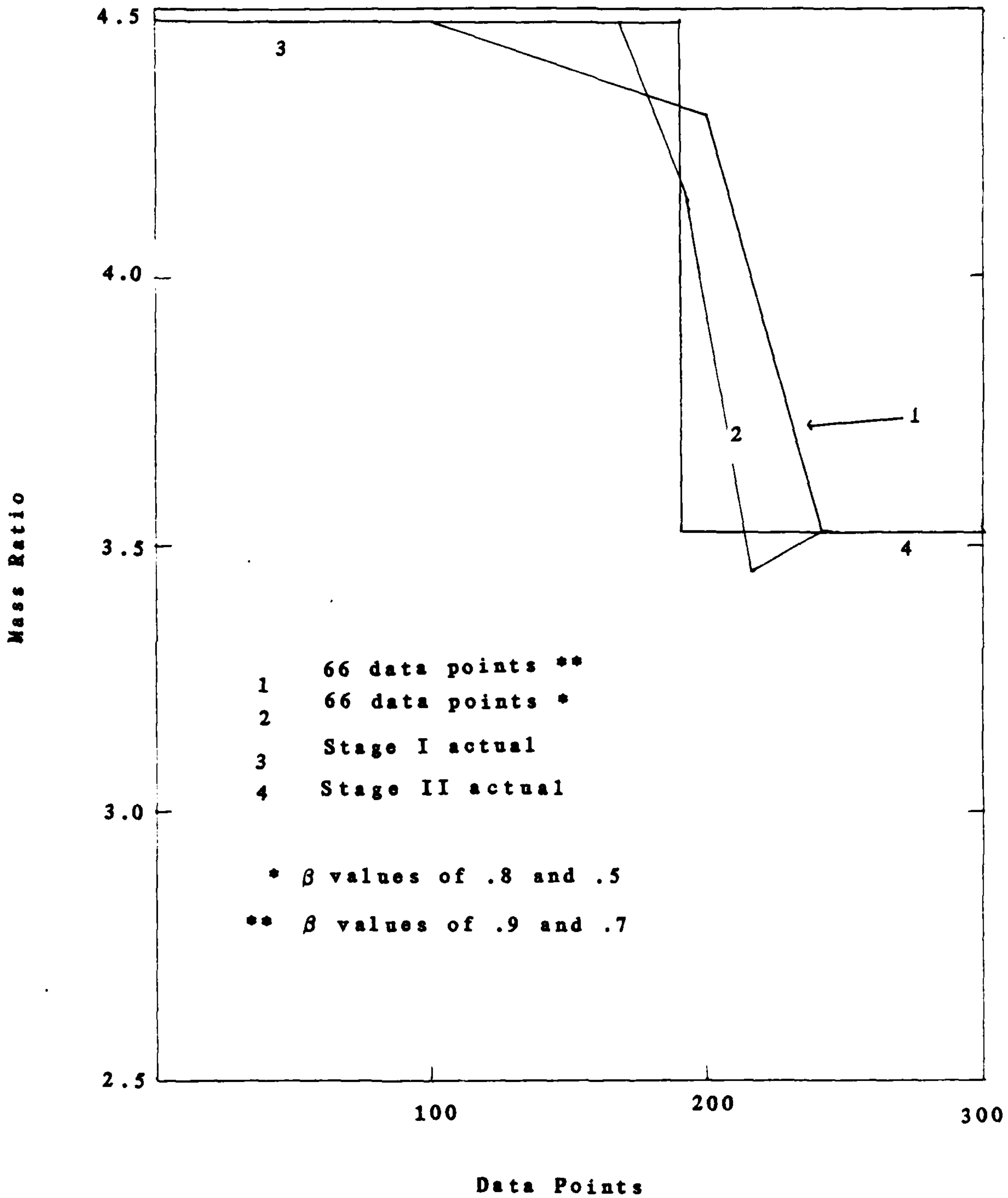


Figure 12. Test Cases: 66, 66

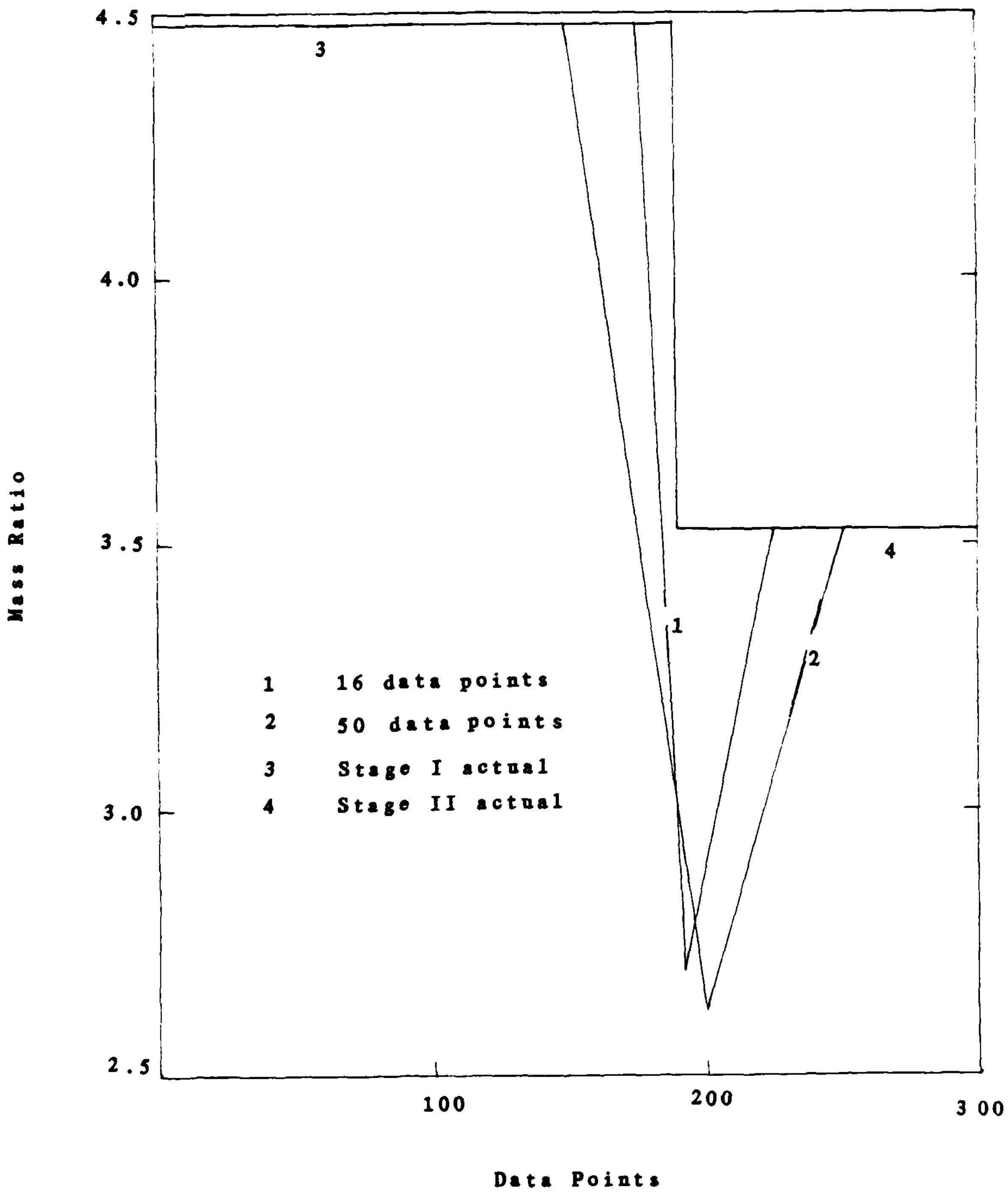


Figure 13. Test Cases: 16, 50

poorest performance are shown in Figures 9 and 14.

It was determined that the filter was only able to observe the staging event successfully when the event occurred within one of the Bayes loop segments of data. In addition, the staging event needed to occur in the latter portion of the data, and the more stage I data that was present, the better the performance of the filter. Table 17 lists the summary of the data segments which were run.

Table 17. Summary of Data Points

# of points	Staging event datasegment	# of Stg I points	# of Stg II points	Ratio StgI/StgII
16	176-192	14	2	7
24	168-192	22	2	11
32	160-192	30	2	15
48	144-192	46	2	23
50	150-200	40	10	4
64	128-192	62	2	31
66	132-198	58	8	7.25
100	100-200	90	10	9

* Note that the staging event occurs at point # 190

Notice that the smallest ratio that successfully converged was the 50 point case. Ratios of less than 4 did not converge. This was apparently due to the fact that the estimator could not change the performance parameters instantaneously, as the stage effectivly does. Therefore, it seemed to need a good memory (the previous stage's data), along with a little new data, to enable a partial correction of the data. On the next data segment which was processed, the entire set of stage II data enabled the further

refinement of the estimate, and in most cases, was very close to the actual value.

A further correlation existed between the ratio and the performance of the estimator. Careful examination of the figures, and the information in Table 17 shows that the higher ratio numbers produced the most accurate results. The case which used 64 data points for each segment very closely approximated the actual data for both exhaust velocity, and mass ratio, and it had the highest Stg I/Stg II ratio. This further supports the contention that the filter needed to process the data slightly differently during the staging event in order to assure success.

Examination of the figures, and the data for the test cases run with 66 data points indicate a trend with the β values. The case which included β values of .9 and .7 seemed to perform a little better than the case with β equal .8 and .5 . This tends to indicate that the filter needs to keep a reasonable amount of memory to correctly estimate the performance parameters. The value at .7 does this, allowing for just enough change to incorporate the variations caused by the staging event.

The figures and the data show that the filter estimated the performance parameters very well during most of the stages thrusting. As the staging event was approached however, each of the cases diverged, proceeded to initiate an estimate of the second stage's performance, and then

converged on the second stage's performance.

Conclusions

A method for the estimation of launch vehicle performance parameters and the position and velocity vectors has been examined. The specific parameters were exhaust velocity and mass ratio. The data obtained from the test cases shows that the filter will estimate the performance parameters. The occurrence of the staging event within a segment of data did not appear to present a problem as long as the event occurred significantly towards the end of a particular data segment. This provided a means by which the filter could perform a staged change from one state to another.

Recommendations for additional work could be accomplished by using data already generated at this point. The added features could include a form of residual monitoring in the program which would watch for a quadratic departure in the in-track position residuals. This would suggest the probability that the time was close to a staging event. The program could then shift the data segment so that the staging event would always occur towards the latter portion of the data. This also suggests the possibility for iterating back over the staging event to try and refine the guess. Another change could include an alteration of the accuracy of the measurements. These could be altered to determine if there is a limit for the resolution of the radar

and infared system that must be obtained to allow for convergence. Finally, additional values could be input for the β values to see if additional changes would alter the results obtained, especially with the smaller data segment size (10-30). The overall objective, however, of observing exhaust velocity and mass ratio was successful.

```
      print*,'y=',y(1,nxt)*6378.145d+ØØ,y(2,nxt)*6378.145d
     +      +ØØ,y(3,nxt)*6378.145d+ØØ,y(4,nxt)*7.9Ø536828d+ØØ
      print*,'y=',y(5,nxt)*7.9Ø536828d+ØØ,y(6,nxt)*7.9Ø536828
     +      d+ØØ,y(7,nxt),y(8,nxt)

      write(17,2)

ccccc begin loop to integrate

      do 1Ø incc= Ø,nit
          call haming(nxt)
          inb= inb + 1
          do 3Ø ind=1,3
              r(ind)= y(ind,nxt)
              v(ind)= y(ind+3,nxt)
3Ø          continue
          call razel(r,v,rho,az,el,to,t,rs,trm,ino)
          if (ioh.ne.1Ø) then
              print*,'do you want noisy data,y or n'
              read*,type
              if (type.eq.'y') then
                  print*,'input a seed number from 1-21483647d+ØØ'
                  read*,dseed
                endif
              ioh= 1Ø
            endif
          if (type.eq.'y') then
              rho= rho + ggnqf(dseed)*sigrho
              az= az + ggnqf(dseed)*sigaz
              el= el + ggnqf(dseed)*sigel
            endif
          write(14,*) rho,az,el,t

          if (inb.eq.1Ø) then
              do 34 ins=1,3
                  rm(ins) = r(ins)*6378.145d+ØØ
                  vm(ins) = v(ins)*25936.2647d+ØØ
34              continue
            call mag(r)
            call mag(rm)
            call mag(vm)
            call mag(v)
            dot = r(1)*v(1)+r(2)*v(2)+r(3)*v(3)
            gamma = dacos(dot/(r(Ø)*v(Ø)))
            tm= t*8Ø6.8118744d+ØØ
            write(17,6) rm(Ø),vm(Ø),gamma/rad,tm,y(7,nxt),y(8,nxt)
            inb= Ø
          endif
        if (incc.eq.nit-1) then

cccccc        print out final data

              print*,'the final values for r and v are='
              print*,'y=',y(1,nxt)*6378.145d+ØØ,y(2,nxt)*6378.145d+ØØ
     +                ,y(3,nxt)*6378.145d+ØØ,y(4,nxt)*7.9Ø536828d+ØØ
              print*,'y=',y(5,nxt)*7.9Ø536828d+ØØ,y(6,nxt)*7.9Ø536828
     +            d+ØØ,y(7,nxt),y(8,nxt)
              write(17,12)
              write(17,4) (y(inf,nxt),inf=1,8),t
              write(17,*)
            endif
1Ø      continue
      endfile(unit=17)
      endfile(unit=14)
```

```
          print*,'input the launch point number'
          read*,lp
          if (lp.eq.0) then
              llat=53.7d+00*rad
              llon=158.2d+00*rad
            endif
          if (lp.eq.1) then
              llat=43.5d+00*rad
              llon=132.0d+00*rad
            endif
          if (lp.eq.2) then
              llat=1.0d+00*rad
              llon=1.0d+00*rad
            endif
          if (lp.eq.3) then
              llat=1.0d+00*rad
              llon=1.0d+00*rad
            endif
          lrs(0)=1.0d+00
          call lstime(llst,t,to,llon)
          ans= '1'
          call radst(lrs,llat,llst,t,to,ans,ino)
          ino=0
          print*,'input initial velocity in ft/s'
          read*,ivel
          ivel= ivel/25936.24764d+00
          do 40 inr=1,3
40            lvv(inr)=ivel*lrs(inr)
          print*,'how much do you want to nudge the velocity'
          read*,nudge
          lvv(3)= lvv(3)+lvv(3)*nudge
          do 44 inw= 1,3
               y(inw,nxt)= lrs(inw)
               y(inw+3,nxt)= lvv(inw)
44          continue
        endif

      tp= tp/806.8118744d+00
      print*,'input the number of iterations'
      read*,nit
      dt=tp/nit
      nxt=0

cccccc  write initial header data

      write(17,*) 'The initial state vector for the missile is'
      write(17,12)

      inb= 0
      dseed= 38888.d+00
      sigrho= .00001d+00
      sigaz= .001d+00
      sigel= .001d+00
      type= 'n'

cccccc  initialize haming and reset the time

      nxt = 0
      call haming(nxt)
      t= tepoch
      if (nxt.eq.0) stop
      write(17,4) (y(inf,nxt),inf=1,8),t
      print*,y(1,nxt),y(2,nxt),y(3,nxt),y(4,nxt),y(5,nxt),y(6,nxt),
     +       y(7,nxt),y(8,nxt)
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c       This program obtains the truth model data for the
c       problem of the launch vehicle. The program is set up
c       for several different types of vehicles.
c
c       Capt. Dave Vallado  1984
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

cccccc  the common terms

        common /ham/ t,y(72.4),f(72.4),err(72),n,dt.mode
        double precision t,y,f,err,dt
        integer n,nxt,mode

cccccc  all the other variables

        integer nit.ina,inb.incc,ind,ine,inf,ino
        double precision sigaz,tepoch,lp,llat,llon,lrs(Ø:3).ivel,
     +     llst,rad.tp,rho,az,el,to,dseed,sigrho,lvv(Ø:3),nudge,
     +     r(Ø:3),v(Ø:3),rs(Ø:3),rcv(Ø:3),trm(3,3),sigel,
     +     rm(Ø:3),vm(Ø:3),dot,gamma,tm
        character type,ans,typed

cccccc  begin the program

2      format(6x,'r  (km)',8x,'v (ft/sec)',6x,'gamma  (deg)',6x,
     +       'time (sec)',4x,'ve  (DU/TU)',4x,'m    ')
4      format(2x,4e2Ø.13,/,2x,4e2Ø.13,/,2x,'the initial time is ',
     +       f6.4)
6      format(2(1x,f14.6),4(2x,f14.1Ø))
12     format(9x,'x',12x,'y',12x,'z',1Øx,'xdot',9x,'ydot',9x,'zdot'.
     +       1Øx,'ve',1Øx,'M')

        open(unit=17.file='product'.access='sequential',status='new')
        open(unit=14.file='tdata',access='sequential',status='new')

        rad= 3.14159265359d +ØØ/18Ø.Ød +ØØ
        nxt=1
        mode= Ø
        n= 8

cccccc  input initial data

        print*,'input the length of the flight in seconds, and time'
        read*,tp,tepoch
        t = tepoch

        print*,'input state vector, or have it calculated,y or n'
        read*,typed

        if (typed.eq.'y') goto 5
        if (typed.eq.'n') goto 7

5       print*,'input the intial state vector for the vehicle'
        read*,y(1,nxt),y(2,nxt),y(3.nxt),y(4,nxt),y(5,nxt),y(6,nxt).
     +       y(7,nxt),y(8,nxt)

7       if (typed.eq.'n') then
            print*,'launch site points are as follows:'
            print*,'Ø  Petropavlovsk  53.7N  158.2E'
            print*,'1  Vlaidivostok   43.5N  132.ØE'
            print*,'2  Turyantunum                 '
            print*,'3  Plesek                      '
```

PROGRAM TLAN.F

Description

This program accomplishes the numerical integration check for the ICBM launch trajectory. The basic operation is very similar to the operation of tn.f, however, the program is set up to calculate the launch point for the ICBM, of which several choices are available. The program then calculates the initial velocity for the missile, assumed to be staright up from the local coordinate system, and then proceeds to nudge the vehicle over so the gravity turn can be executed. The numerical integration by Haming is identical, and instead of one period being integrated, the user can input how long, in seconds, the trajectory is to be integrated.

The variables and usage are almost identical to the program tn.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
cccccc this subroutine calculates the lst of the site     cccccccccccc
       subroutine lstime(lst,t,to,lon)

       double precision lst,t,to,lon

       double precision thtgo,twopi,gst

       to= 0.0d +00
       twopi= 6.28318530718d +00
       thtgo= 98.85481d +00*(3.14159265359d +00/180.0d +00)
       gst= thtgo + 1.00273790093d +00*((t*1.44686457d+00/
     +      1440.0d +00) - to)
       gst= dmod(gst,twopi)
       lst= gst + lon
       lst= dmod(lst,twopi)

       return
       end

c      that should do it!!!!!!!!!!!!!!!!
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

       include '/en/en84d/dvallado/dhaming'
       include '/en/en84d/dvallado/rhstru'
```

```
      if (rhovec(1).eq.Ø.Ød+ØØ) then
          if (rhovec(2).gt.Ø.Ød+ØØ) az= 9Ø.Ød+ØØ*rad
          if (rhovec(2).lt.Ø.Ød+ØØ) az= 27Ø.Ød+ØØ*rad
          if (rhovec(2).eq.Ø.Ød+ØØ) then
              az= Ø.Ød+ØØ
              if (rhovec(3).gt.Ø.Ød+ØØ) el= 9Ø.Ød +ØØ*rad
              if (rhovec(3).lt.Ø.Ød+ØØ) el= -9Ø.Ød+ØØ*rad
            endif
      endif
      if ((rhovec(1).ne.Ø.Ød+ØØ).and.(rhovec(2).ne.Ø.Ød+ØØ)) then
          az= datan(rhovec(2)/rhovec(1))
          el= datan(rhovec(3)/dsqrt(rhovec(1)*rhovec(1) + rhovec(2)*
     +                  rhovec(2)))
          if (rhovec(1).lt.Ø.Ød+ØØ) az= az + 18Ø.Ød +ØØ*rad
          if ((rhovec(1).gt.Ø.Ød+ØØ).and.(rhovec(2).lt.Ø.Ød+ØØ))  az=
     +               az + 36Ø.Ød+ØØ*rad
        endif
      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c     this subroutine calculates the position vector of the site     c
cccccc        for either a land or space based system           ccccc
      subroutine radst(rs,lat,lst,t,to,ans)

      double precision rs(Ø:3),lat,lst,t,to
      character ans

      double precision sta,ste,sti,stomga,stargp,stv(Ø:3),stm,stn,
     +    stnuo
      integer ino

cccccc land based sensor
      if (ans.eq.'l') then
          if (ino.eq.Ø) then
              print*,'input the elevation of the site'
              read*,rs(Ø)
              ino= 1Ø
            endif

          rs(1)= rs(Ø)*dcos(lat)*dcos(lst)
          rs(2)= rs(Ø)*dcos(lat)*dsin(lst)
          rs(3)= rs(Ø)*dsin(lat)
          return
        endif

cccccc space based sensor
      if (ans.eq.'s') then
          if (ino.eq.Ø) then
              print*,'input the tracking sat orbit data, a e i,w,w'
              read*,sta,ste,sti,stomga,stargp
            endif
          stn= dsqrt(1/(sta*sta*sta))
          stm= stn*(t-to)
          call randv(sta,ste,sti,stomga,stargp,stnuo,stm,rs,stv)

          call mag(rs)
          if (ino.eq.Ø) then
64            format(3x,'a',6x,'e',6x,'i',5x,'omega',3x,'argp',4x,'m')
66            format(6(1x,f6.3))
              write(17,*)
              write(17,*) 'the tracking satellite data is'
              write(17,64)
              write(17,66) sta,ste,sti,stomga,stargp,stm
              ino= 1Ø
            endif
        endif
      return
      end
```

```
      end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
cccccc  this subroutine calculates rho az and el for a given r and v ccc
      subroutine razel(r,v,rho,az,el,to,t,rs,trm)

      double precision r(Ø:3),v(Ø:3),rho,az,el,to,t,rs(Ø:3),trm(3,3)

      double precision lat,lon,lst,zvec(Ø:3),svec(Ø:3),evec(Ø:3),rad,
     +   rhove(Ø:3),kvec(Ø:3),rhovec(Ø:3),re(Ø:3),rse(Ø:3)
      integer ing,inh,ini,inj,ink,inl,inm,inn
      character ans

      if (ing.eq.Ø) then
          print*,'enter sensor type, land or space, in quotes'
          read*,ans
          ing= 1Ø
          rad= 3.14159265359d +ØØ/18Ø.Ød+ØØ
          kvec(1)= Ø.Ød+ØØ
          kvec(2)= Ø.Ød+ØØ
          kvec(3)= 1.Ød+ØØ
        endif
      if ((ans.eq.'l').and.(inh.eq.Ø)) then
          print*,'input the lat and lon of site in deg, east+, west-'
          read*,lat,lon
          lat= lat*rad
          lon= lon*rad
          inh= 1Ø
        endif
        call lstime(lst,t,to,lon)
        call radst(rs,lat,lst,t,to,ans)

        do 1ØØ ini=1,3
            rhove(ini)= r(ini) - rs(ini)
1ØØ       continue
        call mag(rhove)
        rho= rhove(Ø)
ccccc   set up local coordinate system
        do 11Ø inj= 1,3
            zvec(inj)= rs(inj)/rs(Ø)
11Ø       continue
        call cross(kvec,zvec,evec)
        do 112 inm=1,3
            evec(inm)= evec(inm)/evec(Ø)
112       continue
        call cross(evec,zvec,svec)
        do 114 inn= 1,3
            svec(inn)= svec(inn)/svec(Ø)
114       continue
cccccc  Set up the transformation for IJK = trm SEZ

        do 12Ø inl=1,3
            trm(inl,1)= svec(inl)
            trm(inl,2)= evec(inl)
            trm(inl,3)= zvec(inl)
12Ø       continue
        do 121 innl=1,3
            re(innl)= r(innl)
            rse(innl)= rs(innl)
121       continue
ccccc  Convert to SEZ for calculations

      do 13Ø ink= 1,3
          rhovec(ink)= rhove(1)*trm(1,ink) + rhove(2)*trm(2,ink)
     +                + rhove(3)*trm(3,ink)
ccccccc   NOTE!!!!!!!   here we do NOT transform r to SEZ
ccccccc   since we will not be calculating H as in obser !!!!
          rs(ink)= rse(1)*trm(1,ink) + rse(2)*trm(2,ink)
     +                + rse(3)*trm(3,ink)
13Ø     continue
```

```
cccccc                  print out final data
                        print*,'the final values for r and v are='
                        print*,'r=',y(1,nxt),y(2,nxt),y(3,nxt)
                        print*,'v=',y(4,nxt),y(5,nxt),y(6,nxt)
                        write(17,2)
                        write(17,4) (y(inf,nxt),inf=1,6)
                        write(17,*)
                        write(17,*)
                      endif
                 endif
1Ø             continue
          endfile(unit=17)
          endfile(unit=14)
          end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                     c
cccccc    this subroutine calculates r and v given the orbit elements cccc
          subroutine randv(a,e,inc,omega,argp,nuo,m,r,v)

          double precision a,e,inc,omega,argp,nuo,m,r(Ø:3),v(Ø:3)

          double precision rad,p,e1,eØ,mo

          rad= 3.1415926535d +ØØ/18Ø.Ød +ØØ
          mo= m*rad
          inc= inc*rad
          argp= argp*rad
          omega= omega*rad
          p= a*(1-e*e)
cccccc    newton rhapson iteration
          e1= mo
8         eØ= e1
          e1=eØ-(eØ-e*dsin(eØ)-mo)/(1.Ød +ØØ - e*dcos(eØ))
          if (dabs(e1-eØ).gt.1.Ød -12) then
             e1=eØ-(eØ-e*dsin(eØ)-mo)/(1.Ød +ØØ - e*dcos(eØ))
             print*,'e1=',e1
             go to 8
            endif

cccccc    find the value of the true anomaly

          nuo= datan2((dsqrt(1.Ød +ØØ -e*e))*dsin(e1)/(1.Ød +ØØ -e*
     +         dcos(e1)),(e-dcos(e1))/(e*dcos(e1)-1.Ød +ØØ))
cccccc    position and velocity vectors

          r(1)= p*dcos(nuo)/(1.Ød +ØØ + e*dcos(nuo))
          r(2)= r(1)*dtan(nuo)
          r(3)= Ø.Ød +ØØ
          v(1)= -dsin(nuo)/dsqrt(p)
          v(2)= (e+dcos(nuo))/dsqrt(p)
          v(3)= Ø.Ød +ØØ

          return
          end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                     c
cccccc    this subroutine calculates the magnitude of a vector   ccccccccc
          subroutine mag(rx)

          double precision rx(Ø:3)

          rx(Ø)= dsqrt(rx(1)*rx(1)+rx(2)*rx(2)+rx(3)*rx(3))
          return
          end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                     c
cccccc    this subroutine calculates the cross product of 2 vectors   cccc
          subroutine cross(rin,vin,vx)

          double precision rin(Ø:3),vin(Ø:3),vx(Ø:3)

          vx(1)= rin(2)*vin(3)-vin(2)*rin(3)
          vx(2)= -rin(1)*vin(3)+vin(1)*rin(3)
          vx(3)= rin(1)*vin(2)-vin(1)*rin(2)
          call mag(vx)
          return
```

```
cccccc  calculate initial angular momentum and specific mech energy

        call mag(r)
        call mag(v)
        angm= dsqrt(a*(1-e*e))
        se= v(Ø)*v(Ø)/2.Ød +ØØ - 1.Ød +ØØ/r(Ø)
        tp= tp*13.44686457d +ØØ

cccccc  write initial header data

        write(17,12)
        write(17,6)a.e,inc,omega.argp.nuo.mo.tp.itn
        write(17,5)
        write(17,7)se,angm
        write(17,2)

        ina= Ø
        inb= Ø
        dseed= 38888.d+ØØ
        sigrho= .ØØØØ1d+ØØ
        sigaz= .ØØ1d+ØØ
        sigel= .ØØ1d+ØØ
        type= 'n'

cccccc  initialize haming and reset the time

        call haming(nxt)
        t= Ø.Ød+ØØ
        write(17,4) (y(inf,nxt),inf=1,6)

cccccc  begin loop to integrate

        do 1Ø incc= Ø,nit
            call haming(nxt)
            inb= inb + 1
            do 3Ø ind=1,3
                r(ind)= y(ind,nxt)
                v(ind)= y(ind+3,nxt)
3Ø            continue
            call razel(r,v,rho,az,el,to,t,rs,trm)

            if (ioh.ne.1Ø) then
                print*,'do you want noisy data,y or n'
                read*,type
                if (type.eq.'y') then
                    print*,'input a seed number from 1-21483647d+ØØ'
                    read*,dseed
                  endif
                ioh= 1Ø
              endif
            if (type.eq.'y') then
                rho= rho + ggnqf(dseed)*sigrho
                az= az + ggnqf(dseed)*sigaz
                el= el + ggnqf(dseed)*sigel
              endif
            write(14,*) rho,az,el,t
            if ((itn.ne.5Ø).and.(inb.eq.1Ø)) then
                ina= ina + 1
                if (ina.eq.1Ø) then
                    call mag(r)
                    call mag(v)
                    se= v(Ø)*v(Ø)/2.Ød +ØØ - 1.Ød +ØØ/r(Ø)
                    call cross(r,v,rcv)
                    angm= rcv(Ø)
                    write(17,7)se,angm
                    ina= Ø
                    print*,'the rho az el time is',rho,az/rad,el/rad
                  endif
                inb=Ø
                if (incc.lt.nit-1Ø) then
                    write(17,4) (y(ine,nxt),ine=1,6)
                  endif
                if (incc.eq.nit-1) then
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c       This program checks out the numerical integrator for
c       haming. It does this by an integration of an orbit, once
c       around the orbit.
c
c       Capt. Dave Vallado  1984
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

cccccc  the common terms

        common /ham/ t,y(72,4),f(72,4),err(72),n,dt,mode
        double precision t,y,f,err,dt
        integer n,nxt,mode

cccccc  all the other variables

        integer nit,ina,inb,incc,ind,ine,inf
        double precision a,e,inc,omega,argp,m,tm11,tm12,tm21,tm22,
     +          tm31,tm32,rad,se,angm,tp,rho,az,el,to,dseed,sigrho,
     +          r(Ø:3),v(Ø:3),rs(Ø:3),rcv(Ø:3),trm(3,3),sigel,sigaz
        character type

cccccc  begin the program

2       format(9x,'x',14x,'y',14x,'z',12x,'xdot',12x,'ydot',12x,'zdot')
4       format(6(1x,f14.11))
5       format('the specific mech energy and ang momentum are')
6       format(7(1x,f6.3),1x,f8.3,1x,i6)
7       format(2(8x,f14.11))
12      format(3x,'a',6x,'e',6x,'i',5x,'omega',3x,'argp',4x,'nuo',4x,'m'
     +       ,4x,'period',3x,'# it')

        open(unit=17,file='product',access='sequential',status='new')
        open(unit=14,file='tdata',access='sequential',status='new')

        print*,'input the data a,e,i,w,w,m for the orbit'
        read*,a,e,inc,omega,argp,m
        call randv(a,e,inc,omega,argp,nuo,m,r,v)

cccc    convert from PQW to IJK

        rad= 3.14159265359d +ØØ/18Ø.Ød +ØØ
        nxt=1
        t= Ø.Ød +ØØ
        to= Ø.Ød +ØØ
        mode= Ø
        n= 6

        tm11= dcos(omega)*dcos(argp)-dsin(omega)*dsin(argp)*dcos(inc)
        tm12= -dcos(omega)*dsin(argp)-dsin(omega)*dcos(argp)*dcos(inc)
        tm21= dsin(omega)*dcos(argp)+dcos(omega)*dsin(argp)*dcos(inc)
        tm22= -dsin(omega)*dsin(argp)+dcos(omega)*dcos(argp)*dcos(inc)
        tm31= dsin(argp)*dsin(inc)
        tm32= dcos(argp)*dsin(inc)

        y(1,nxt)= tm11*r(1)+tm12*r(2)
        y(2,nxt)= tm21*r(1)+tm22*r(2)
        y(3,nxt)= tm31*r(1)+tm32*r(2)
        y(4,nxt)= tm11*v(1)+tm12*v(2)
        y(5,nxt)= tm21*v(1)+tm22*v(2)
        y(6,nxt)= tm31*v(1)+tm32*v(2)

        print*,'the initial value for r and v ='
        print*,'r=',y(1,nxt),y(2,nxt),y(3,nxt)
        print*,'v=',y(4,nxt),y(5,nxt),y(6,nxt)

cccccc  start program

        tp= 2.Ød +ØØ*3.1415926535d +ØØ*sqrt(a*a*a)
        print*,'input the number of iterations'
        read*,nit
        dt=tp/nit
        nxt=Ø
```

RANDV

This subroutine calculates the position and velocity vectors given the initial orbit data. The algorithm uses the eccentric anomaly calculation, and the Newton Rhapson iteration to find the mean anomaly. Transformations from the Perifocal coordinate system are used to convert the result into the IJK frame.

MAG

This subroutine simply calculates the magnitude of a vector.

CROSS

This subroutine calculates the cross product of 2 vectors.

RAZEL

This subroutine calculates the range, azimuth and elevation for the the truth model and leastsquares, and Bayes filter programs. It uses the next 2 subroutines that are described to accomplish this. Note that 2 seperate versions are used, one shown with the numerical integrator tn.f, and the other shown with obser. The difference here is the inclusion of a transformation to IJK, which is documented in the subroutine.

RADST

This subroutine calculates the position vector of the site. For a land based site, the user is asked to input latitude and longitude in degrees. The elevation is input in DU's. If the site is a satellote, the user is asked to input the orbit parameters for the calculation of the orbit. Notice that for any follow on effort, it would be advisabler to incorporate a seperate time to the satellite observer so that the satellite could be positioned over the launch point. The program, as written now will place the satellite at the same local sidereal time, no matter what orbit parameters are used.

LSTIME

This subroutine calculates the local sidereal time for the site. Notice that the value is input in degrees for 1984.

```
the haming common
rad        radians to degrees conversion
dseed      input # to IMSL routine for random numbers
sigrho     range
sigaz      azimuth      standard deviation
sigel      elevation
```

Remaining Variables

```
r(0:3)     Position vector
v(0:3)     Velocity vector
tm11, ...  Transformation matrix from PQW to IJK
se         Specific Mechanical Energy
angm       Angular Momentum
tp         Time period  (TU's)
rho        Range
az         Azimuth
el         Elevation
rcv(3)     r cross v
trm(3,3)   Transformation from SEZ to IJK
rad        degree to radian conversion
```

t_o initial time

```
Counters and misc Holders
ina,inb,incc,ind,ine,inf,ioh
```

Subroutines used

```
randv
mag
cross
razel
radst
1stime
dhaming  (See Appendix B)
rhstru   (See Appendix C)
```

Notes

mode = 0 so EOM only

time step is critical for obtaining convergence on the orbit

starting the integration at perigee is difficult since the vehicle is moving the fastest.

APPENDIX A

PROGRAM TN.F

Description

This program checks out the numerical integrator that is programmed in Haming. It accomplishes this by integrating the two-body equation, once around it's orbit.

The subroutine randv, takes input orbit elements and converts them to position and velocity vectors in the PQW system (reference 1). A Newton Rhaphson iteration is employed to convert the mean anomaly to the eccentric anomaly. This is then input to find the true anomaly, from which the postion and velocity vectors are readily obtained.

The main program then rotates the postion and velocity vectors to the IJK frame and assigns these values to the state vector y. The program then calculates the period of the orbit and divides the iteration as 1/500th of the period. Haming is then called to iontegrate the orbit around, with the only stops being to caluculate the specific mechanical energy and the angular momentum from the position and velocity vectors at 10 step intervals. The main data is placed in the system file with only minimal input directed to the screen.

User Inputs

a	Semi Major axis (DU)
e	eccentricity
i	inclination (deg)
Omega	Longitude of ascending node (deg)
Argp	Argument of Perigee (deg)
M	Mean anomaly (deg)
itn	number of iterations
'l', 's'	Land or space based sensor
'y', 'n'	Type, whether or not you want noisy data

If land based

Lat	Lat of the site (deg)
Lon	Longitude of the site (deg)
rs(0)	Elevation of the site (DU)

If space based

sta	tracker semi major axis (DU)
ste	tracker eccentricity
sti	tracker inclination (deg)
stomga	tracker long. of ascending node (deg)
stargp	tracker argument of perigee (deg)

Variables To be Set

BIBLIOGRAPHY

1. Bate, Roger R., Mueller, Donald D., and White, Jerry E. Fundamentals of Astrodynamics. New York : Dover Publications, Inc., 1971.

2. Gross, Donald W., "Estimation of Launch Vehicle Performance Parameters from Two Orbiting Sensors", Masters Thesis. Wright Patterson AFB, Ohio : Air Force Institute of Technology, December 1982.

3. Miller, Capt. Gregory D., "Estimation of Launch Vehicle Performance Parameters from an Orbiting Sensor", Masters Thesis. Wright Patterson AFB, Ohio : Air Force Institute of Technology, December 1981.

4. Sutton, George P., and Ross, Donald M., " Rocket Propulsion Elements ", New York: Wiley Publishing, 1976.

5. U.S. Space Launch Systems (u), Navy Space Systems Activity. Report No. NSSA-R-20-72-2. P.O. Box 92960, Worldway Postal Center, Los Angeles, California, 90009.

6. Wiesel, William E., Jr. Lecture material distributed in MC731, Modern Methods of Orbit Determination. School of Engineering, Air Force Institute of Technology, Wright Patterson AFB, Ohio, 1984.

7. Wie~el, William E., Jr. Lecture Material distributed in MC533, Problems in Spaceflight. School of Engineering, Air Force Institute of Technology, Wright Patterson AFB, Ohio, 1984.

APPENDIX B

SUBROUTINE HAMING

Description

This program is a fourth order differential equations integrator. It carries four copies of the state vector along, and extrapolates them to find the next value. It then corrects this answer to find the new value of the state vector.

To use the predictor-corrector algorithm, an initial state vector must be stored in y(*,1). nxt is then set to 0 and one call is made to haming to initialize the EOM EOV etc. The time is then reset to the epoch, and normal use can proceed, as long as nxt does not still = 0 upon exit from haming. This would mean that haming was unable to use the initial state vector as a guess. (maybe the step size is too big)

User Inputs

none

Variables to be Set

t	independent variable time
dt	time step
y(*,1)	1 copy of the state vector to be input
n	number of equations to be integrated
errest	estimate of truncation error (generally not used)
mode	0 - EOM only 1 - EOM and EOV
nxt	sets the transitions between Haming and Rhs

(Collectively I call these variables The Haming Common)

Remaining Variables

f(*,4)	4 copies of the equations of motion. Rhs updates these on each call from haming.
tol	a tolerance parameter
hh	step size holder
xo	time holder

Counters and misc holders,
ida, idb, idc, idd, ide, idf, idg, idh, idi, idj, idk, idl, idn

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine haming(nxt)

c     Haming is an ordinary differential equations integrator
c     that is a fourth order predictor-corrector algorithm
c     which means that it carries along the last 4
c     values of the state vector, and extrapolates these
c     values to obtain the next value (the prediction part)
c     and then corrects the extrapolated value to find a
c     new value for the state vector.
c
c     The value nxt specifies which of the 4 values
c     of the state vector is the "next" one, and it is
c     updated automatically.  To use, an initial state
c     vector must be stored in y(*,1). nxt is then set to Ø
c     and one call is made to haming to initialize the
c     EOM and EOV, etc. The time is then reset to the initial
c     time and normal use can begin as long as nxt not Ø.
c     nxt = Ø means that haming was unable to converge.
c
c     The user supplies the external routine rhs(nxt) which
c     evaluates the equations of motion.
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c     Common terms and variable declarations

      common /ham/ t,y(72,4),f(72,4),errest(72),n,dt,mode
      double precision t,y,f,errest,dt
      integer n,mode,nxt

      integer ida,idb,idc,idd,ide,idf,idg,idh,idi,idj,idk,idl,idm,idn
      double precision tol,hh,xo

c     the variables are used as follows
c     t              independent variable  (time)
c     y(72,4)        state vector in 4 copies, nxt points to next one
c     f(72,4)        equations of motion, 4 copies
c                           call rhs(nxt) updates entry in f
c     errest         estimate of truncation error
c     n              number of equations being integrated
c     dt              time step
c     mode           Ø for EOM, 1 for EOM and EOV

      tol = 1.Ød-12
      if(nxt) 19Ø,1Ø,2ØØ

c     switch on strating algorithm or normal propagation
c     this is hamings starting algorithm....a predictor - corrector
c     needs 4 values of the state vector, and you only have 1, the
c     I.C.  Haming uses a pricard iteration (slow and painfull) to get
c     the other 3.
c     if it fails, nxt will be Ø on exit, otherwise, nxt=1, and it's ok.

1Ø    xo = t
      hh = dt/2.Ød+ØØ
      call rhs(1)
      do 4Ø ida = 2,4
          t = t + hh
          do 2Ø idb = 1,n
2Ø            y(idb,ida) = y(idb,ida-1) + hh*f(idb,ida-1)
          call rhs(ida)
          t = t + hh
          do 3Ø idc = 1,n
3Ø            y(idc,ida) = y(idc,ida-1) + dt*f(idc,ida)
4Ø      call rhs(ida)
      idd = -1Ø
5Ø    ide = 1
      do 12Ø idf = 1,n
          hh = y(idf,1) + dt*(9.Ød+ØØ*f(idf,1) + 19.Ød+ØØ*f(idf,2)
     +         - 5.Ød+ØØ*f(idf,3) + f(idf,4))/24.Ød+ØØ
          if (dabs(hh-y(idf,2)).lt.tol) goto 7Ø
          ide = Ø
```

```
70          y(idf,2) = hh
            hh = y(idf,1)+dt*(f(idf,1)+4.0d+00*f(idf,2)+f(idf,3))/3.0d+00
            if (dabs(hh-y(idf,3)).lt.tol) goto 90
            ide = 0
90          y(idf,3) = hh
            hh= y(idf,1) + dt*(3.0d+00*f(idf,1) + 9.0d+00*f(idf,2) +
     +          9.0d+00*f(idf,3) + 3.0d+00*f(idf,4))/8.0d+00
            if (dabs(hh-y(idf,4)).lt.tol) goto 110
            ide = 0
110         y(idf,4) = hh
120       continue
        t = xo
        do 130 idg = 2,4
            t = t + dt
130       call rhs(idg)
        if (ide) 140,140,150
140     idd = idd + 1
        if (idd) 50,280,280
150     t = xo
        ide = 1
        idd = 1
        do 160 idh = 1,n
160         errest(idh) = 0.0
        nxt = 1
        go to 280
190     idd = 2
        nxt = iabs(nxt)

cccccc  this is hamings normal propagation loop -

200     t = t + dt
        idl = mod(nxt,4) + 1
        go to (210,230),ide

cccccc  permute the index nxt modulo 4

210     go to (270,270,270,220),nxt
220     ide = 2
230     idi = mod(idl,4) + 1
        idj = mod(idi,4) + 1
        idk = mod(idj,4) + 1

cccccc      this is the predictor part

        do 240 idm = 1,n
            f(idm,idl)= y(idm,idl) + 4.0d+00*dt*(2.0d+00*f(idm,idk)-
     +          f(idm,idj) - 2.0d+00*f(idm,idi))/3.0d+00
240         y(idm,idl) = f(idm,idi) - 0.925619835d+00*errest(idm)

c       now the corrector - fix up the extrapolated state
c       based on the better value of the equations of motion

        call rhs(idl)
        do 250 idn = 1,n
             y(idn,idl)= (9.0d+00*y(idn,idk)-y(idn,idi)+3.0d+00*dt*(
     +            f(idn,idl) + 2.0d+00*f(idn,idk) - f(idn,idj)))/8.0d+00
             errest(idn) = f(idn,idl) - y(idn,idl)
250          y(idn,idl) = y(idn,idl) + 0.07438016553d+00 * errest(idn)
        go to (260,270),idd
260     call rhs(idl)
270     nxt = idl
280     return
        end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

APPENDIX C

SUBROUTINE RHS

Description

This program calculates the equations of motion and equation of variation (the A matrix) for the problem which is evaluated. It serves merely as a data source for Haming and is called using nxt, thus no specific inputs are needed other than those in the common block with Haming.

User Inputs

type of vehicle	parameters
'0'	Satellite in orbit
'1'	Titan IIIB
'2'	Titan IIID
'3'	Thor LV-2F

Variables to be set

The Haming common

Remaining variables

xmu	Gravitational Parameter (1 $^{DU}/_{TU}$)
r32, r52	r^3, r^5
v32	v^3
vel	Vehicle velocity
vat	Combination of vel, acc, t ve and m
vve	(velocity)(Exhaust velocity)
vem	(exhaust velocity)(m)
mass	mdot/initial mass
acc	Vehicle acceleration
am	A matrix
masso	initial mass
mdot	mass flow rate
V_e	Exhaust velocity

Counters and misc holders

ira, irb, irc, ird, ire, irf, irg, irh, iri, irj, irk, ii, jj

Subroutines used

vehd

Notes

3 different versions of this program were run.

rhstru	contained subroutine vehd and is listed here
rhsam	did not contain vehd instead of the call, V_e= y(7,nxt), M= y(8,nxt) printed the a matrix before phidot= a*phi
rhslb	the same as rhs am except the A matrix was not printed

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine rhs(nxt)
c
c     rhs calculates the equations of motion and /or not and
c     the equations of variation for the problem of estimating
c     launch vehicle performance parameters.
c
c     the state vector is split out as
c     y(1-3,nxt) are the x,y,z components of the postion vector
c     y(4-6,nxt) are the x,y,z components of the velocity vector
c     y(7-72,nxt) is the state transition matrix, stored as
c                 columns of phi end to end
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

c     Common terms and varaible declarations
      common /ham/ t,y(72,4),f(72,4),err(72),n,dt,mode
      double precision t,y,f,err,dt
      integer n,mode,nxt

      integer ira,irb,irc,ird,ire,irf,irg,irh,iri,irj,irk
      double precision r32,v32,vel,vat,vve,vem,mass,acc,r52,am(8,8),
     +        masso,mdot,ve

c     this data statement hardwires the parts of the
c     a matrix which are never changed...only the middle
c     3 rows change each time
      do 10 ira=1,8
          do 10 irb=1,3
              am(irb,ira)= 0.0d+00
10      continue
      do 20 irc=1,8
          do 20 ird=7,8
              am(ird,irc)= 0.0d+00
20      continue
      am(1,4)= 1.0d+00
      am(2,5)= 1.0d+00
      am(3,6)= 1.0d+00

c     the basic function of rhs is to calculate the equations
c     motion (the f enrties) from the given current state
c     (stored in y) and the time t
c
c     *******************************************************
c
c          EVALUATE THE EQUATONS OF MOTION
c
c     *******************************************************
c     reference Bates Meuller & White, pg 10, N body problem
c     with origin in sun.
c
c     position dot = velocity vector

      f(1,nxt) = y(4,nxt)
      f(2,nxt) = y(5,nxt)
      f(3,nxt) = y(6,nxt)

cccccc     velocity dot = gravity accel

      r32 = ( y(1,nxt)*y(1,nxt) + y(2,nxt)*y(2,nxt)
     +        + y(3,nxt)*y(3,nxt)   ) ** 1.5d+00
      vel = ( y(4,nxt)*y(4,nxt) + y(5,nxt)*y(5,nxt)
     +        + y(6,nxt)*y(6,nxt)   ) ** .5d+00
      v32= vel ** 3.0d +00
```

```
cccccc     Set the constants which will be used in the A matrix

      xmu= 1.Ød +ØØ
      if (ire.eq.Ø) then
          print*,'input the type of missile to be evaluated'
          print*,' the coices are as follows'
          print*,'Satellite in orbit          Ø'
          print*,'Titan-IIIB                  1'
          print*,'Titan-IIID                  2'
          print*,'Thor-LV-2f short tank       3'
          read*,irf
          ire= 1Ø
        endif
      if (irf.eq.Ø) then
          vve= 1.Ød+ØØ
          vem= 1.Ød+ØØ
          acc=Ø.Ød+ØØ
          mass=1.Ød+ØØ
          goto 6
        endif

      call vehd(ve,mdot,masso,t,irf)

      mass= mdot/masso
      vve= vel*ve
      vem= ve*mass
      acc= ve*mass/(1-mass*t)

        y(7,nxt)= ve

        y(8,nxt)= mass

6     f(4,nxt) = - xmu * y(1,nxt) / r32 + acc*y(4,nxt)/vel
      f(5,nxt) = - xmu * y(2,nxt) / r32 + acc*y(5,nxt)/vel
      f(6,nxt) = - xmu * y(3,nxt) / r32 + acc*y(6,nxt)/vel
      f(7,nxt) = Ø.Ød+ØØ
      f(8,nxt) = Ø.Ød+ØØ

c     end of equations of motion
c     is this all ?

      if( mode .eq. Ø) return

c     it isnt all ... calculate the
c
c     ************************************************
c
c        EQUATIONS OF VARIATION
c
c     *************************************************
c
c     FIRST, calculate a matrix.... only lower 3x3 isnt hard wired

      r52 = r32 **( 5.Ød+ØØ/3.Ød+ØØ )

cccccc     diagonal terms in a matrix

      an(4,1) = -xmu/r32 + 3.Ød+ØØ*xmu*y(1,nxt)*y(1,nxt)/r52
      an(5,2) = -xmu/r32 + 3.Ød+ØØ*xmu*y(2,nxt)*y(2,nxt)/r52
      an(6,3) = -xmu/r32 + 3.Ød+ØØ*xmu*y(3,nxt)*y(3,nxt)/r52

c     off diagonal terms in a matrix
c     use symmetry to avoid as much calculation
c     as possible...this point is deep within lots of loops!!!!
```

```
      am(4,2) = 3.Ød-ØØ*xmu*y(1,nxt)*y(2,nxt)/r52
      am(5,1) = am(4.2)
      am(4,3) = 3.Ød-ØØ*xmu*y(1,nxt)*y(3,nxt)/r52
      am(6,1) = am(4.3)
      am(5,3) = 3.Ød-ØØ*xmu*y(2,nxt)*y(3,nxt)/r52
      am(6,2) = am(5.3)

cccccc     now same stuff for the other terms

      am(4,4)= -y(4,nxt)*y(4,nxt)*acc/v32 + acc/vel
      am(5,5)= -y(5,nxt)*y(5,nxt)*acc/v32 + acc/vel
      am(6,6)= -y(6,nxt)*y(6,nxt)*acc/v32 + acc/vel

      am(4,5)= -y(4,nxt)*y(5,nxt)*acc/v32
      am(5,4)= am(4,5)

      am(4,6)= -y(4,nxt)*y(6,nxt)*acc/v32
      am(6,4)= am(4,6)

      am(5,6)= -y(5,nxt)*y(6,nxt)*acc/v32
      am(6,5)= am(5,6)

      am(4,7)= y(4,nxt)*acc/vve
      am(5,7)= y(5,nxt)*acc/vve
      am(6,7)= y(6,nxt)*acc/vve

      vat= acc*acc*t/vem + acc/mass

      am(4,8)= y(4,nxt)*vat/vel
      am(5,8)= y(5,nxt)*vat/vel
      am(6,8)= y(6,nxt)*vat/vel

c     the a matrix is now calculated
c
c     NOW, calculate phi dot = a * phi and put into last
c     64 slots of f matrix

      do 8ØØ irg = 1.8
         do 8ØØ irh = 1,8
            iri = 8*irh + irg
            f(iri,nxt) = Ø.ØØd+ØØ
            do 7ØØ irj = 1,8
               irk = 8*irh + irj
               f(iri,nxt)= f(iri,nxt) + am(irg,irj)*y(irk,nxt)
 7ØØ        continue
 8ØØ     continue

cccccc     phi dot = a * phi is now done

      return
      end
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                        c
c     this subroutine calculates the launch vehicle data                 c
      subroutine veho(ve,mdot,masso,t,irf)

      double precision ve,mdot,masso,t
      integer irf

      double precision time,isp,thrust

      time= t*13.44686457d+00
      if (irf.eq.1) then
          if (time.lt.1.3d+00) then
              isp= 256.0d+00
              thrust= 434900.0d+00
              masso= 305970.0d+00
            endif
          if ((time.lt.4.0d+00).and.(time.ge.1.3d+00)) then
              isp= 317.0d+00
              thrust= 102300.0d+00
              masso=73816.0d+00
            endif
          if ((time.lt.6.0d+00).and.(time.ge.4.0d+00)) then
              isp= 292.0d+00
              thrust= 16000.0d+00
              masso= 14676.0d+00
            endif
        endif

      if (irf.eq.2) then
          if (time.lt.1.3d+00) then
              isp= 301.0d+00
              thrust= 523000.0d+00
              masso= 307500.0d+00
            endif
          if ((time.lt.4.0d+00).and.(time.ge.1.3d+00)) then
              isp= 317.0d+00
              thrust= 102300.0d+00
              masso=73670.0d+00
            endif
          if ((time.lt.6.0d+00).and.(time.ge.4.0d+00)) then
              isp= 444.0d+00
              thrust= 30000.0d+00
              masso= 36122.0d+00
            endif
          if ((time.lt.10.0d+00).and.(time.ge.6.0d+00)) then
              isp= 284.0d+00
              thrust= 15000.0d+00
              masso= 2721.0d+00
            endif
        endif
      if (irf.eq.3) then
          if (time.lt.2.5d+00) then
              isp= 251.0d+00
              thrust= 170000.0d+00
              masso= 106092.0d+00
            endif
          if ((time.lt.8.0d+00).and.(time.ge.2.5d+00)) then
              isp= 290.0d+00
              thrust= 10000.0d+00
              masso= 1743.7d+00
            endif
        endif

      ve= isp/806.818744d+00
      mdot= thrust/ve

      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
```

APPENDIX D

PROGRAM TNA.F

Description

This program checks out the A matrix as described in chapter 3. The inputs are simply the state vector y(*,1), which can either be left unchanged, or re-input by the user at run time. The program then calls rhs which calculates the A matrix directely and prints it out to a seperate file. Then, using equation (4-1) the A matirx is again calculated using the different method. The results are printed out to the same output file, and the results can then be compared.

User inputs

'y' or 'n'   ans- whether or not the user wants to change the given state vector.
If yes, input the new state vector y(8)

Variables to be set

the haming common

Remaining variables

```
xu(8,nxt)       unperturbed state vector
fu(8,nxt)       unperturbed F matrix
amat(8,8)       A matrix
delta           delta of each iteration
Counters and misc holders
iaa,iab,iac,iad,iae,iaf,iag,iah
```

Subroutines used

rhsam (See Appendix C)

Notes

watch using nxt in and out of dhaming, it is not always equal to 1.

THE A MATRIX

$$
\left[
\begin{array}{cccc}
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\dfrac{-\mu}{r^3} + \dfrac{3\mu x^2}{r^5} & \dfrac{3\mu xy}{r^5} & \dfrac{3\mu xz}{r^5} & \dfrac{-v_x{}^2 a}{v^3} + \dfrac{a}{v} \\
\dfrac{3\mu yx}{r^5} & \dfrac{-\mu}{r^3} + \dfrac{3\mu y^2}{r^5} & \dfrac{3\mu yz}{r^5} & \dfrac{-v_y v_x a}{v^3} \\
\dfrac{3\mu xz}{r^5} & \dfrac{3\mu yz}{r^5} & \dfrac{-\mu}{r^3} + \dfrac{3\mu z^2}{r^5} & \dfrac{-v_x v_z a}{v^3} \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{array}
\right.
$$

$$
\left.
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
\dfrac{-v_x v_y a}{v^3} & \dfrac{-v_x v_z a}{v^3} & \dfrac{v_x a}{v V_e} & \dfrac{v_x}{v}\left(\dfrac{a^2 t}{V_e m} + \dfrac{a}{M}\right) \\
\dfrac{-v_y{}^2 a}{v^3} + \dfrac{a}{v} & \dfrac{-v_y v_z a}{v^3} & \dfrac{v_y a}{v V_e} & \dfrac{v_y}{v}\left(\dfrac{a^2 t}{V_e m} + \dfrac{a}{M}\right) \\
\dfrac{-v_y v_z a}{v^3} & \dfrac{-v_z{}^2 a}{v^3} + \dfrac{a}{v} & \dfrac{v_z a}{v V_e} & \dfrac{v_z}{v}\left(\dfrac{a^2 t}{V_e m} + \dfrac{a}{M}\right) \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{array}
\right]
$$

where

$$A_{ij} = \partial F_i / \partial \bar{x}_j$$

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c       This program checks out the A matrix for the problem of
c       estimation of launch vehicle performance parameters.  It
c       does this by having rhs calculate the A matrix from an
c       input state vector.  Then, each element of the state vector
c       is perturbed, and the columns of A are calculated by
c       subtracting the original F matrix, from the perturbed
c       F matrix and dividing by delta.
c
c       Capt. Dave Vallado  1984
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

        common /ham/ t,y(72,4),f(72,4),errest(72),n,dt,mode
        double precision t,y,f,errest,dt
        integer n,mode,nxt

        integer iaa,iab,iac,iad,iae,iaf,iag,iah
        double precision  xu(8,1),fu(8,8),amat(8,8),delta
        character ans

4       format(8(1x,e12.6))
        open(unit=16,file='aprod',access='sequential',status='new')
        nxt=1
        mode =1
        n= 72
        t= .4967?4413d+ØØ
        dt= Ø.Ød+ØØ

cccccc initialize the state vector

        y(1,nxt)= 2.464Ø282763d +ØØ
        y(2,nxt)= .1995Ø378763d +ØØ
        y(3,nxt)= .1995Ø378763d +ØØ
        y(4,nxt)= -.Ø7137664494d +ØØ
        y(5,nxt)=  .44435648672d +ØØ
        y(6,nxt)=  .44435648672d +ØØ
        y(7,nxt)= Ø.Ød +ØØ
        y(8,nxt)= Ø.Ød +ØØ

        print*,'the current y-12345678 values are',(y(iaa,nxt),
     +        iaa=1,8)
        print*,'do you want to change? y or n in quotes'
        read*,ans
        if (ans.eq.'n') goto 5

        print*,'input the new state vector,1 to 8,and the time'
        read*,(y(iab,nxt),iab=1,8),t

5       write(16,*)'the A matrix check data is as follows for '
        write(16,*)'the initial state vector y of'
        write(16,*) (y(iac,1),iac=1,8),t

         write(16,*)

cccccc call rhs and have the A matrix printed out

        call rhs(nxt)

cccccc set initial state and f vectors

        do 8 iad= 1,8
            xu(iad,nxt)= y(iad,nxt)

            fu(iad,nxt)= f(iad,nxt)
8         continue
```

```
ccccc perturb each element

      write(16,*)'now perturb each element of the state vector'
      mode=0
      do 10 iae= 1,8
          delta= -y(iae,nxt)*.0001d +00
          if (dabs(delta).lt.1.0d-14) goto 9
          y(iae,nxt) = delta + y(iae,nxt)
          call rhs(nxt)
          do 9 iaf= 1,8
              amat(iaf,iae)= (f(iaf,nxt)-fu(iaf,nxt))/delta
9           continue
          y(iae,nxt)= xu(iae,nxt)
10      continue

      write(16,4) ((amat(iag,iah),iah=1,8),iag=1,8)
      endfile(unit=16)
      end

      yee haw!!!!!!!!!!!!!!!!!
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      include '/en/en84d/dvallado/rhsam'
```

APPENDIX E

PROGRAM TNPH.F

Description

This program accomplishes much the same function as tna.f. It really only mechanizes the discussion in chapter 3. The input is a state vector y(8) which can be altered by the user, however, one should note that the time step is based on an orbit from the family where a=2.5DU. The program calls haming which numerically integrates the state through about $^1/_5$ of the orbit. The program then prints out the o matrix to a file called phprod. Then, one by one, the states are perturbed, translated through time, and reinitialized until equation 4-2 has been used to successively calculate the columns of the o matrix. The results of the second calculation are then output to the same file for comparision.

User inputs

'y' or 'n' ans- whether or not you want to change the given state vector

If yes, input the new state vector

Variables to be set

the haming common
tp time period

Remaining Variables

xu(8)	unperturbed state vector
cphi(8,8)	calculated o matrix
xut(8)	unperturbed state vector, moved through time
delta	delta on each iteration

Counters and misc holders
ipa, ipb, ipc, ipd, ipe, ipf, ipg, iph, ipi, ipj, ipk, ipl, ipm, ipn, ipo, ipp, ipq, ipr, ips, ipt, ipu, ipv

Subroutines used

dhaming (See Appendix B)
rhslb (See Appendix C)

Notes

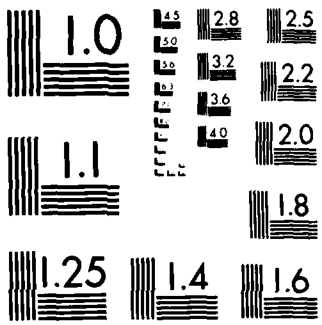
be sure to set n=72, t=0, dt=
Remember that a= 2.5 DU's !!
must reset o and state on each iteration
watch values of nxt in and out of the subroutine calls

```
:cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

     This program checks out the Phi matrix for the problem of
     estimation of launch vehicle performance parameters. It
     does this by having rhs calculate the Phi matrix.  Then,
     each element of the input state vector is perturbed, and
     moved through time. The columns of the Phi matrix are
     calculated by subtracting the orginal state, from the
     perturbed state, and dividing by delta.

     Capt. Dave Vallado  1984
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

     common /ham/ t,y(72,4),f(72,4),errest(72),n,dt,mode
     double precision t,y,f,errest,dt
     integer n,mode,nxt

     double precision  xu(8,1),cphi(8,8),xut(8,1),tp,delta,tstart
     integer ipa,ipb,ipc,ipd,ipe,ipf,ipg,iph,ipi,ipj,ipk,ipl,ipm,
    +        ipn,ipo,ipp,ipq,ipr,ips,ipt,ipu,ipv
     character ans

     format(3(1x,f12.6))
     open(unit=15,file='phprod',access='sequential',status='new')

     n=72
     mode=1
     nxt=1
     tstart= .496729413d +00

:cc  initialize the state vector

     y(1,nxt)=  2.48402827630d +00
     y(2,nxt)=  0.19950378763d +00
     y(3,nxt)=  0.19950378763d +00
     y(4,nxt)=  -.07137664494d +00
     y(5,nxt)=   .44435648672d +00
     y(6,nxt)=   .44435648672d +00
     y(7,nxt)=  0.0d +00
     y(8,nxt)= 0.0d +00
     tp= 2.0d +00*3.1415962535d +00*dsqrt(15.625d+00)

     print*,'the current values of y -12345678 are',(y(ipcc,nxt),
    +     ipcc= 1,8)
     print*,'do you want to change the values?'
     read*,ans
     if (ans.eq.'n') goto 5

     print*,'input new state vector,time and tp for tp/500'
     read*,(y(ipa,nxt),ipa=1,8),tstart,tp

:ccc set unperturbed state vector

     do 6 ipb=1,8
         xu(ipb,nxt)= y(ipb,nxt)
       continue

     dt=tp/500.0d +00
     t= tstart

:ccc initialize phi matrix

     do 7 ipc= 9,72

         y(ipc,nxt)= 0.0d +00
       continue
     do 8 ipd= 9,72,9
         y(ipd,nxt)= 1.0d +00
       continue

     write(15,*) 'this data is from the state vector y ='
     write(15,*) (y(ipw,nxt),ipw=1,8),t
     write(15,*) 'the initial phi matrix is'
     write(15,4) ((y(ipe,nxt),ipe=ipf,72,8),ipf=9,16)
```

AD-A151 927 ESTIMATION OF LAUNCH VEHICLE PERFORMANCE PARAMETERS(U) 2/2
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL
OF ENGINEERING D A VALLADO DEC 84 AFIT/GA/AA/84D-10

UNCLASSIFIED F/G 12/1 NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

```
ccccccc propogate unperturbed state vector through time

        nxt=0

ccccccc initialize haming and reset time

        call haming(nxt)
        t=tstart
        do 9 ipg=1,50
            call haming(nxt)
            if (ipg.eq.50) then
                print*,'the last iteration of the phi matrix is'
                print*,((y(iph,nxt),iph=ipi,72,8),ipi=9,16)
              endif
9         continue

cccccc  set unperturbed, moved through time, state vector

        do 10 ipj=1,8
            xut(ipj,1)= y(ipj,nxt)
10        continue

        write(15,*) 'phi is as follows after 50 iterations'
        write(15,4) ((y(ipk,nxt),ipk=ipl,72,8),ipl=9,16)

cccccc  now perturb each one of the elements of the state vector

        do 20 ipm=1,8

cccccc      reset the state and phi and perturb one element

            do 21 ipr=1,8
                y(ipn,1)= xu(ipn,1)
21            continue
            do 22 ipo= 9,72
                y(ipo,1)= 0.0d +00
22            continue
            do 24 ipp= 9,72,9
                y(ipp,1)= 1.0d +00
24            continue
            nxt= 0
            t= tstart
            delta= - xu(ipm,1)*.0001d +00
            if (abs(delta).lt.1.0d -14) goto 20
            print*,'delta=',delta
            y(ipm,1)= delta+xu(ipm,1)
            call haming(nxt)
            t=tstart

cccccc      move the perturbed vector through time, and cal phi

            do 26 ipq=1,50
                call haming(nxt)
26            continue
            do 28 ipr=1,8
                cphi(ipr,ipm)= (y(ipr,nxt) - xut(ipr,1))/delta
28            continue
20        continue

        write(15,*) 'calcu phi is as follows after 50 iterations'
        write(15,4) ((cphi(ipt,ips),ips=1,8),ipt=1,8)
        endfile(unit=15)
        print*,'the calculated phi is'
        print*,((cphi(ipv,ipu),ipu=1,8),ipv=1,8)

        end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

        include '/en/en84d/dvallado/dhaming'
        include '/en/en84d/dvallado/rhslb'
```

APPENDIX F

PROGRAM TNH.F

Description

This program checks out the H matrix for the specific problem. It does this by use of equation 4-3 in chapter 3. The program starts by inputting an initial state vector which the user can change at run time. Then, similar to the A matrix check done in tna.f, the program has obser calculate the H matrix directly, and then calculates the columns individually by perturbing each element in the state vector, and dividing out the resulting differences in the calculated G matricis.

User Inputs

'y' or 'n'   ans- whether or not the given state vector should be changed

If yes, input the new state vector

Variables to be set

the haming common

Remaining Variables

xu(8)	unperturbed state vector
hm(3,8)	H matrix
zpredu(3)	unperturbed G matrix
zpred	G matrix
delta	delta on each iteration

Counter and misc holders
iha, ihb, ihc, ihd, ihe, ihf, ihg, ihh, ihi, ihj

Subroutines used

obser, razel (See Appendix G)

randv, mag, cross, radst, lstime (See Appendix A)

G MATRIX

$$G = \begin{pmatrix} rho \\ az \\ el \end{pmatrix} = \begin{pmatrix} \sqrt{x^2 + y^2 + z^2} \\ \tan^{-1}(y/x) \\ \tan^{-1}\left(z/\sqrt{x^2 + y^2}\right) \end{pmatrix}$$

H MATRIX

$$\delta \begin{pmatrix} rho \\ az \\ el \end{pmatrix} = H \; \delta \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\delta \begin{pmatrix} rho \\ az \\ el \end{pmatrix} = \underbrace{\frac{\partial \begin{pmatrix} rho \\ az \\ el \end{pmatrix}}{\partial \begin{pmatrix} S \\ E \\ Z \end{pmatrix}}}_{A} \; \underbrace{\frac{\partial \begin{pmatrix} S \\ E \\ Z \end{pmatrix}}{\partial \begin{pmatrix} I \\ J \\ K \end{pmatrix}}}_{B} \; \delta \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$A = \left[\begin{array}{ccc|c} \frac{x}{(x^2+y^2+z^2)^{1/2}} & \frac{y}{(x^2+y^2+z^2)^{1/2}} & \frac{z}{(x^2+y^2+z^2)^{1/2}} & ..0.. \\ \frac{-y/x^2}{1+(y/x)^2} & \frac{1/x}{1+(y/x)^2} & 0 & ..0.. \\ \frac{-xz/(x^2+y^2)^{3/2}}{1+z^2/(x^2+y^2)} & \frac{-yz/(x^2+y^2)^{3/2}}{1+z^2/(x^2+y^2)} & \frac{1/(x^2+y^2)^{1/2}}{1+z^2/(x^2+y^2)} & ..0.. \end{array}\right]$$

$$B = \left[\begin{array}{c|c|c} \hat{S} & \hat{E} & \hat{Z} \end{array}\right]$$

$$= \left[\; \frac{\hat{E} \times \hat{Z}}{|\hat{E} \times \hat{Z}|} \quad \frac{\hat{k} \times \hat{Z}}{|\hat{k} \times \hat{Z}|} \quad \frac{\bar{r}s}{|\bar{r}s|} \;\right]$$

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c
c       This program checks out the H matrix for the problem of
c       estimation of launch vehicle performance parameters.  It
c       does this by having rhs calculate the H matrix from an
c       input state vector.  Then, each element of the state vector
c       is perturbed, and the columns of H are calculated by
c       subtracting the original G matrix, from the perturbed
c       G matrix and dividing by delta.
c
c       Capt. Dave Vallado   1984
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

        common /ham/ t,y(72,4),f(72,4),errest(72),n,dt,mode
        double precision t,y,f,errest,dt
        integer n,mode,nxt

        integer nxt,mode,iha,ihb,ihc,ihd,ihe,ihf,ihg,ihh,ihi,ihj
        double precision  xu(8,1),zpredu(3),hm(3,8),zpred(3),delta
     +        ,h(3,8),ql(3,3),tob
        character ans

4       format(8(1x,e12.6))
        open(unit=13,file='hprod',access='sequential',status='new')
        nxt=1
        dt= Ø.Ød +ØØ
        t= 4.91762119d+ØØ
        mode =1

cccccc initialize the state vector

        y(1,nxt)= .8Ø2359Ø2456d +ØØ
        y(2,nxt)= 1.674249Ø8479d +ØØ
        y(3,nxt)= 1.674249Ø8471d +ØØ
        y(4,nxt)= -.59899756237d +ØØ
        y(5,nxt)=  .14353Ø34570d +ØØ
        y(6,nxt)=  .14353Ø34570d +ØØ
        y(7,nxt)= Ø.Ød +ØØ
        y(8,nxt)= Ø.Ød +ØØ

        print*,'the current y-12345678 values are',(y(iha,nxt),
     +        iha=1,8)
        print*,'do you want to change? y or n in quotes'
        read*,ans
        if (ans.eq.'n') goto 5

        print*,'input the new state vector,1-8, and time'
        read*,(y(ihb,nxt),ihb=1,8),t

5       write(13,*)'the H matrix check data is as follows for '
        write(13,*)'the initial state vector y of'
        write(13,*) (y(ihc,1),ihc=1,8),t
        write(13,*)


cccccc call obser and calculate G and H


        tob= t
        call obser(tob,ql,zpred,h,nxt)

        write(13,*) 'obser calculates H as follows '
        do 7 ihi= 1,3

            write(13,4) (h(ihi,ihj),ihj=1,8)
7         continue

cccccc set initial state and g vectors

        do 8 ihd= 1,8
            xu(ihd,nxt)= y(ihd,nxt)
8         continue
        do 6 ihk=1,3
            zpredu(ihk)= zpred(ihk)
6         continue
```

```
cccccc perturb each element

       mode=0
       do 10 ihe= 1,8
           delta= -y(ihe,nxt)*.0001d +00
           y(ihe,nxt) = delta + y(ihe,nxt)
           call obser(tob,ql,zpred,h,nxt)
           if (abs(delta).lt.1.0d -14) goto 10
           do 9 ihf= 1,3
               hm(ihf,ihe)= (zpred(ihf)-zpredu(ihf))/delta
9            continue
           y(ihe,nxt)= xu(ihe,nxt)
10       continue

cccccc  Print out the result

        write(13,*)
        write(13,*) 'the perturbed calculation for H is as follows'
        write(13,4) ((hm(ihg,ihh),ihh=1,8),ihg=1,3)

      write(13,*)

       endfile(unit=13)
       end

c      yee haw!!!!!!!!!!!!!!!!!!
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

       include '/en/en84d/dvallado/obser'
```

APPENDIX G

SUBROUTINE OBSER

Description

This program calculates the observation relationships. The main function is to calculate the G matrix, the predicted data, and the H matrix which is simply the partial of G with respect to the state vector. I have called the G matrix in the derivations by G and z, and the obser program uses zpred and z. They are the same thing.

User inputs

none

Variables to be set

The Haming Common

tob	time of each observation
trm(3,3)	transformation from SEZ to IJK
to	initial time

Remaining Variables

qi	Q inverse	
zpred(3)	G matrix	
h(3,8)	H matrix	
r(3)	position vector	
v(3)	velocity vector	
rho	range	
az	azimuth	
el	elevation	
rs(3)	site vector	
sigrho	range	
sigaz	azimuth	standard deviation
sigel	elevation	

Counters and misc holders
ioa, ioc, iod, iof, iou, iov, iow, ohm1, azdnom, eldnom, ioe, elbtm, hit(3,3)

Subroutines used

razel
randv
mag
cross (See Appendix A)
radst
lstime

Notes

do not transform r to SEZ as in tn.f

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      subroutine obser (tob,q1,zpred,h,nxt)

c     this program calculates the following data
c     zpred          observation matrix
c     h              H matrix
c     q1             q inverse
c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      double precision tob,q1(3,3),zpred(3),h(3,8)
      integer nxt

      common /ham/ t,y(72,4),f(72,4),err(72),n,dt,mode
      double precision t,y,f,err,dt,
      integer n,mode

      double precision to,r(0:3),v(0:3),rho,az,el,rs(0:3),sigel,ioe
     +    ,ohm1,azdnom,eldnom,elbtm,hit(3,3),trm(3,3),sigrho,sigaz
      integer ioa,ioc,iod,iof,iot,iou,iov,iow

c     *******************************************************
c
c      RANGE - AZIMUTH - ELEVATION  DATA
c
c     *******************************************************
c
c     q inverse matrix
c
      do 10 ioc= 1,3
          do 10 iod= 1,3
              q1(ioc,iod)= 0.0d +00
10      continue
c      print*,'input the sigma rho, az el'
c      read*,sihrho,sigaz,sigel
       sigrho= .00001d+00
       sigaz= .001d+00
       sigel= .001d+00
       q1(1,1)= sigrho*sigrho
       q1(2,2)= sigaz*sigaz
       q1(3,3)= sigel*sigel
       ioe= q1(1,1)*q1(2,2)*q1(3,3)
       do 12 iof= 1,3
          q1(iof,iof)= q1(iof,iof)/ioe
12       continue
      to= 0.0d +00
      do 11 ioa=1,3
          r(ioa)= y(ioa,nxt)
          v(ioa)= y(ioa+3,nxt)
11      continue

      call razel(r,v,rho,az,el,to,t,rs,trm)

ccccc this calculates the G matrix

      zpred(1)= rho
      zpred(2)= az
      zpred(3)= el

cccccc  The H matrix
cccccc  note, the r and rs are in SEZ

      ohm1= (r(1)-rs(1))*(r(1)-rs(1)) + (r(2)-rs(2))*(r(2)-rs(2))

     +        + (r(3)-rs(3))*(r(3)-rs(3))

      h(1,1) = (1.0d-00/dsqrt(ohm1))*(r(1)-rs(1))
      h(1,2) = (1.0d-00/dsqrt(ohm1))*(r(2)-rs(2))
      h(1,3) = (1.0d-00/dsqrt(ohm1))*(r(3)-rs(3))
      h(1,4) = 0.0d+00
      h(1,5) = 0.0d+00
      h(1,6) = 0.0d+00
      h(1,7) = 0.0d+00
      h(1,8) = 0.0d+00
```

```
      azdnom= 1.Ød+ØØ + ((r(2)-rs(2))/(r(1)-rs(1)))*
     +  ((r(2)-rs(2))/(r(1)-rs(1)))

      h(2,1) = (-(r(2)-rs(2))/((r(1)-rs(1)) * (r(1)-rs(1))))/azdnom
      h(2,2) = (1.Ød-ØØ / (r(1)-rs(1)))/azdnom
      h(2,3) = Ø.Ød -ØØ
      h(2,4) = Ø.Ød -ØØ
      h(2,5) = Ø.Ød -ØØ
      h(2,6) = Ø.Ød -ØØ
      h(2,7) = Ø.Ød -ØØ
      h(2,8) = Ø.Ød -ØØ

      elbtm= (r(1)-rs(1))*(r(1)-rs(1)) + (r(2)-rs(2))*(r(2)-rs(2))
      eldnom= 1.Ød+ØØ + ((r(3)-rs(3))*(r(3)-rs(3)))/elbtm

      h(3,1) = ((-(r(1)-rs(1))*(r(3)-rs(3)))/dsqrt(elbtm*elbtm*elbtm))
     +          / eldnom
      h(3,2) = ((-(r(2)-rs(2))*(r(3)-rs(3)))/dsqrt(elbtm*elbtm*elbtm))
     +          / eldnom
      h(3,3) = (1.Ød+ØØ / dsqrt(elbtm)) / eldnom
      h(3,4) = Ø.Ød +ØØ
      h(3,5) = Ø.Ød -ØØ
      h(3,6) = Ø.Ød -ØØ
      h(3,7) = Ø.Ød -ØØ
      h(3,8) = Ø.Ød +ØØ

ccccc Convert to IJK

      do 2Ø1Ø iot=1,3
          hit(1,iot)= h(1,iot)
          hit(2,iot)= h(2,iot)
          hit(3,iot)= h(3,iot)
2Ø1Ø    continue
      do 2Ø2Ø iou=1,3
          do 2Ø2Ø iov=1,3
          h(iou,iov) = Ø.Ød +ØØ
          do 2Ø2Ø iow=1,3
              h(iou,iov)= hit(iou,iow)*trm(iov,iow) + h(iou,iov)
2Ø2Ø    continue
      return
      end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
cccccc  this subroutine calculates rho az and el for a given r and v ccc
       subroutine razel(r,v,rho,az,el,to,t,rs,trm)

       double precision r(Ø:3),v(Ø:3),rho,az,el,to,t,rs(Ø:3),trm(3,3)

       double precision lat,lon,lst,zvec(Ø:3),svec(Ø:3),evec(Ø:3),rad,
     +    rhove(Ø:3),kvec(Ø:3),rhovec(Ø:3),re(Ø:3),rse(Ø:3)

       integer ing,inh,ini,inj,ink,inl,inm,inn
       character ans

       if (ing.eq.Ø) then
           print*,'enter sensor type, land or space, in quotes'
           read*,ans
           ing= 1Ø
           rad= 3.14159265359d +ØØ/18Ø.Ød+ØØ
           kvec(1)= Ø.Ød+ØØ
           kvec(2)= Ø.Ød+ØØ
           kvec(3)= 1.Ød+ØØ
         endif
       if ((ans.eq.'l').and.(inh.eq.Ø)) then
           print*,'input the lat and lon of site in deg, east+, west-'
           read*,lat,lon
           lat= lat*rad
           lon= lon*rad
           inh= 1Ø
         endif
         call lstime(lst,t,to,lon)
         call radst(rs,lat,lst,t,to,ans)
```

```
        do 100 ini=1,3
            rhove(ini)= r(ini) - rs(ini)
100       continue
        call mag(rhove)
        rho= rhove(0)
        do 110 inj= 1,3
            zvec(inj)= rs(inj)/rs(0)
110       continue
        call cross(kvec,zvec,evec)
        do 112 inm=1,3
            evec(inm)= evec(inm)/evec(0)
112       continue
        call cross(evec,zvec,svec)
        do 114 inn= 1,3
            svec(inn)= svec(inn)/svec(0)
114       continue
cccccc  Set up the transformation for IJK = trm SEZ

        do 120 inl=1,3
            trm(inl,1)= svec(inl)
            trm(inl,2)= evec(inl)
            trm(inl,3)= zvec(inl)
120       continue
        do 121 innl=1,3
            re(innl)= r(innl)
            rse(innl)= rs(innl)
121       continue
ccccc  Convert to SEZ for calculations

      do 130 ink= 1,3
          rhovec(ink)= rhove(1)*trm(1,ink) + rhove(2)*trm(2,ink)
     +               + rhove(3)*trm(3,ink)
          r(ink)= re(1)*trm(1,ink) + re(2)*trm(2,ink)
     +               + re(3)*trm(3,ink)
          rs(ink)= rse(1)*trm(1,ink) + rse(2)*trm(2,ink)
     +               + rse(3)*trm(3,ink)
130     continue

      if (rhovec(1).eq.0.0d+00) then
          if (rhovec(2).gt.0.0d+00) az= 90.0d+00*rad
          if (rhovec(2).lt.0.0d+00) az= 270.0d+00*rad
          if (rhovec(2).eq.0.0d+00) then
              az= 0.0d+00
              if (rhovec(3).gt.0.0d+00) el= 90.0d +00*rad
              if (rhovec(3).lt.0.0d+00) el= -90.0d+00*rad
            endif
      endif
      if ((rhovec(1).ne.0.0d+00).and.(rhovec(2).ne.0.0d+00)) then
          az= datan(rhovec(2)/rhovec(1))
          el= datan(rhovec(3)/dsqrt(rhovec(1)*rhovec(1) + rhovec(2)*
     +               rhovec(2)))
          if (rhovec(1).lt.0.0d+00) az= az + 180.0d +00*rad
          if ((rhovec(1).gt.0.0d+00).and.(rhovec(2).lt.0.0d+00))  az=
     +              az + 360.0d+00*rad
        endif
      return
      end
```

APPENDIX H

PROGRAM LSTSQ.F

Description

This program checks out the initil runs for a least squares estimation of the input problem. The cases used for trial runs consisted of the satellite orbits which were numerically integrated around one orbit. The estimator works by mechanizing the summary shown on the following page. Basically, the data is input, along with the truth model data that is run from program tn.f seperately. After an initial state vector is input, the initial guess, the least squares program takes the guess along with the observation data and tries to estimate the true intital state.

User inputs

tepoch	initial starting time
$\bar{x}$ref(8)	initial guess for the state vector
maxit	number of iterations that least squares will run through while trying to estimate the state
nob	number of observations to be read.
trop	rank of p matrix (used in inversions)

Variables to be set

The Haming Common

Remaining variables

timeob( )	time, rho, az and el for the storage
rho( )	of the truth model data
az( )	
el( )	
phi(8,8)	ϕ
h(3,8)	H matrix
tmat(3,8)	T matrix
tob	holds time of each observation
z(3)	holds G matrix values, each observation
zpred(3)	holds G matrix values, calculated from xref in the program
dx(8,1)	state vector corrections
q1(3,3)	Q^{-1}
resid(3)	residual vector
work	storage for IMSL inverse routine
htq1(8,3)	$T^TQ^{-1}T$
pinv(8.8)	P^{-1}
htq1r(8,1)	$T^TQ^{-1}\bar{r}$
xref(8)	state vector which gets updated

p(8,8)	P matrix (covariance)
tepoch	Initial start time
tmatt(8,3)	T transpose

Subroutines used

mmpy	
mtrans	
mprint	
eigen	
dhaming	(See Appendix B)
rhslb	(See Appendix C)
obser, razel	(See Appendix G)
randv, mag, cross, radst, lstime	(See Appendix A)

Notes

The value of trop was important in calculating the eigenvlaues and eigenvectors. The satellite had a rank of 6, whereas the ICBM had a rank of 8. Difficulties were encountered with using trop=8, so it was left at 6

MMPY

This subroutine multiplies 2 matrices together, and outputs the result. Note that the 2 matricies must be declared identically in and out of the routine, and they must both be 2 dimensional, i.e. (0:3,0:4)

MTRANS

This subroutine calculates the transpose of a matrix.

MPRINT

This subroutine prints a matrix.

EIGEN

This subroutine calculates the eigenvalues and eigenvectors for the filter estimation problems. Note that copies are made to pass to the IMSL routines since these routines destroy the original matrix.

Non-Linear Least Squares Algorithm

1. pick $\bar{x}_{ref}(t_o)$, initial guess for state vector

   - Set Q and read in data for all observations (G)

   - initialize
     $\phi = I$
     $P^{-1} = 0$
     $T^T Q^{-1} \bar{r} = 0$

2. for each observation time

   - move $\bar{x}_{ref}(t_o)$ to $\bar{x}_{ref}(t_i)$

     -Haming and rhs do this for x_{ref}, also o

   - calculate predicted data using $\bar{x}_{ref}(t_i)$, zpred obser does this

   - calculate residual $\bar{r}_i = \bar{z}_i - \bar{z}pred_i$

   - calculate H , obser does this

   - calculate $T = H\ \phi$

   - sum $\sum T^T Q^{-1} T$
     - since P^{-1} gets reset each iteration, it is summed up inside the observation loop

   - sum $\sum T^T Q^{-1} \bar{r}$

   - loop back until all the data is processed

3. Calculations

   - $P = [\ T^T Q^{-1} T]^{-1}$

   - $\delta\bar{x} = P\ \ T^T Q^{-1} \bar{r}$

   - update the $\bar{x}_{ref}(t_o) = \bar{x}_{ref}(t_o) + \delta\bar{x}(t_o)$

   - check convergence (See page 23, $\delta\bar{x}(i) < \sqrt{P_{ii}}$)

   - if good, end with $\bar{x}_{ref}(t_o)$

   - if not, begin at start with $\bar{x}_{ref}(t_o)$ and reset

     $\phi = I \ :\ T^T Q^{-1} \bar{r} = 0 \ :\ P^{-1} = 0 \ :\ t = t_o$
     $y(*,1) = \bar{x}_{ref}(t_o)$

```
cccccc print covariance matrix and find eigenvalues

950     format(/,2x,"Covariance Matrix at epoch is:",/,
     +  8(1x,8e14.7,/) )
        print 950,p
        call eigen(pc,trop)

cccccc load new state vector, and reset phi

        do 960 ibr= 1,8
             y(ibr,1)= y(ibr,nxt)
             xref(ibr,1)= y(ibr,nxt)
960        continue
        call meql(xref,8,1,xrefu)

cccccc extract phi matrix in normal form


        do 985 ibv = 1,8
            do 985 ibw = 1,8
985             phi(ibv,ibw) = y(8*ibw+ibv,nxt)

cccccc calculate updated phi matrix at new start time

        call meql(phi,8,8,phic)
        call linv1f(phic,8,8,phiin,0,work,ier)
        call mtrans(phiin,8,8,phit)
        call mmpy(phit,8,8,pinvn,8,pinvol)
        call mmpy(pinvol,8,8,phiin,8,pinvn)
        call mmpy(beta,8,8,pinvn,8,pinvo)
        call mtrans(beta,8,8,betat)
        call meql(pinvo,8,8,pinvo2)
        call mmpy(pinvo2,8,8,betat,8,pinvo)

        tepoch = t
        idone = 0
        print*,'begin next bayes loop'

10000     continue

c--------------  LOOP BACK FOR BAYES FILTER LOOP ---------------------c

        print*,'we did it, success with bayes'
        end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c        this subroutine makes a copy of a  matrix                   c
         subroutine meql(mat7,mat7r,mat7c,mat8)

         double precision mat7(mat7r,mat7c)
         integer mat7r,mat7c

         double precision mat8(mat7r,mat7c)
         integer imf,img

         do 3000 imf= 1,mat7r
             do 3000 img= 1,mat7c
3000             mat8(imf,img)= mat7(imf,img)

         return
         end
```

```
          do 420 ibe=1,8
              do 420 ibf= 1,8
                  pinvn(ibe,ibf)= pinvo(ibe,ibf) + htqlt(ibe,ibf)
420         continue

          call meql(pinvn,8,8,pinvnc)

cccccc    have we just finsihed printing last pass residuals ?

          if( idone .eq. 1) go to 5000

cccccc    data is processed....improve estimate
cccccc    invert matrix H transpose Q inverse H to find
cccccc    covariance P

          call linv1f(pinvnc,trop,8,p,0,work,ier)
          call meql(p,8,8,pc)

cccccc    from matrix ******  dx = P * T transpose Q inverse r

           do 600 ibc=1,8
              xmxref(ibc,1)= xrefu(ibc,1)- xref(ibc,1)
600              continue
            call mmpy(pinvo,8,8,xmxref,1,pndx)
            do 640 ibd=1,8
               pndxph(ibd,1)= pndx(ibd,1)+htqlr(ibd,1)
640          continue
           call mmpy(p,8,8.pndxph,1,dx)

cccccc    add in state corrections..

          do 700 ilv = 1,8
700           xref(ilv,1) = xref(ilv,1) + dx(ilv,1)

cccccc    print iteration, and current guess

720       format(/.2x,"iteration ".i3,/,2x."state corrections"
     +          ,/.2x,4e20.13,/,2x,4e20.13 )
          print 720.ilc,dx
740       format(/.2x,"current reference trajectory state vector at
     +          at epoch:"./,2x,4e20.13,/,2x,4e20.13,/)
          print 740.xref

cccccc    SUCCESS ?????????
cccccc    check convergence

          ifail = 0
          do 800 ilu = 1,8
             if( dabs(dx(ilu,1)).gt.0.1*dsqrt(dabs(p(ilu,ilu))))
     +                 ifail = 1
800        continue
          if(ifail .eq. 0 ) idone = 1

9999    continue
c------ LOOP BACK FOR NEXT ITERATION OF LEAST SQUARES ----------------c

cccccc FAILURE for the least squares !!!!!!!!!!!!

900     format(2x,"maximum iteration limit exceeded
     +         without convergence.")
        print 900
        stop

cccccc SUCCESS for the least squares !!!!!!!!!!!!!

5000    continue

940     format(/.2x,"CONVERGENCE ACHIEVED.",/
     +        .2x,"In nominia Gaussiam trajectorum referentia",/,
     +        2x,"declarium est estimatia",/)
        print 940
        ilnobs = ilnobs + ilnob
```

```
c---------------- OBSERVATION PROCESSING LOOP ------------------------c

           do 1000 ill = ilnobs,ilnob + ilnobs - 1

cccccc        extract each observation

              tob = timeob(ill)
              z(1)= rho(ill)
              z(2)= az(ill)
              z(3)= el(ill)

cccccc        NUMERICALLY INTEGRATE STATE AND PHI TO OBS TIME
c             the number of steps here is equal to 1 since we
cccccc        have dt set exactely the same as the truth data we read

              nstp= 1
              do 100 ilk = 1,nstp
100                call haming(nxt)

cccccc        OBTAIN MATRICES FOR THIS OBSERVATION

              call obser(tob,ql,zpred,h,nxt)

cccccc        MATRIX STUFF - THIS OBSERVATION

              do 120 ill = 1,ndata
120               resid(ill) = z(ill) - zpred(ill)
              if( ill .lt. ilnobs+5) go to 200
              if(( idone .eq. 1).and.(ill.lt.ilnobs+5)) go to 200
              if(( idone .eq. 1).and.(ill.ge.ilnobs+5)) go to 240
              go to 250
200           continue
              print*,'time. res ='.tob,(resid(ilm).ilm=l,ndata)

cccccc        if this is last pass. weve already converged.
cccccc                 so skip matrix calculations

240           if( idone .eq. 1 ) go to 9000
250           continue

cccccc        extract phi matrix in normal form

              do 260 iln = 1,8
                  do 270 ilo = 1,8
270                   phi(iln,ilo) = y(8*ilo+iln,nxt)
260             continue

cccccc        form matrix ******  tmat = h * phi

              call mmpy(h,3,8,phi,8,tmat)
cccccc        form matrix ******  htql = T transpose * Q inverse

              call mtrans(tmat,3,8.tmatt)
              call mmpy(tmatt,8,3,ql,3,htql)

cccccc        form matrix ****** htqlt = T transpose Q inverse  T
cccccc          (sum through the observations)

              do 290 ilp = 1,8
                  do 290 ilq = 1,8
                      do 280 ilr = 1,ndata
280                       htqlt(ilp,ilq)= htqlt(ilp,ilq)+htql(ilp,ilr)
     +                            *tmat(ilr,ilq)
290             continue

cccccc        form matrix ****** htqlr = T transpose Q inverse r
cccccc          (sum through the observations)

              do 150 ils = 1,8
                  do 150 ilt = l,ndata
150                   htqlr(ils,1)= htqlr(ils,1)+htql(ils,ilt)*
     +                                resid(ilt)

9000          continue
1000        continue

c-------- LOOP BACK FOR OBSERVATION LOOP OF LEAST SQUARES -------------c
```

```
cccccc    READ IN OBSERVATIONS

        open(unit=14,file='tdata',access='sequential',status='old')
        rewind(unit=14)
        print*,'input the total number of observations to be read'
        read*,nob
        do 30 ilb = 1,nob
            read (14,*,end=30) rho(ilb),az(ilb),el(ilb),timeob(ilb)
30        continue
        endfile(unit=14)
        ndata= 3
        print 10,xrefu,tepoch,nob,maxit,ilnob,ibloop,trop,
     +      beta(1,1),beta(2,2),beta(3,3),beta(4,4),beta(5,5),
     +      beta(6,6),beta(7,7),beta(8,8)

cccccc set last iteration flag

        idone = 0
        ilnobs = 1
        call meql(xrefu,8,1,xref)
        do 40 ibj=1,8
            do 40 ibi=1,8
40              pinvo(ibj,ibi)= 0.0d+00

c-------------------- BEGIN BAYES FILTER LARGE LOOP -------------------c

        do 10000 ibg= 1,ibloop

c--------- BEGIN ITERATION LOOP - NONLINEAR LEAST SQUARES -------------c

        dt = timeob(2) - timeob(1)
        do 9999 ilc = 1,maxit

cccccc      REINITIALIZE NUMERICAL INTEGRATION PARAMETERS

            t = tepoch
            mode = 1
            n = 72

cccccc      ics are new reference traj guess

                do 50 ild = 1,8
                    y(ild,1) = xref(ild,1)
50                continue

cccccc      phi initial conditions

            do 60 ile = 9,72
60              y(ile,1) = 0.0d+00
            do 70 ilf = 9,72,9
70              y(ilf,1) = 1.0d+00

cccccc      initialize haming and reset the time

            nxt = 0
            call haming(nxt)
            t= tepoch

cccccc      INITIALIZE BUFFERS FOR MATRIX PRODUCT ACCUMULATION

            do 80 ilg = 1,8
                htqlr(ilg,1) = 0.0d+00
                do 80 ilh = 1,8
80                  htqlt(ilg,ilh) = 0.00d+00

cccccc      print first or last pass redidual headers when necessary

90          format(2x,"First Pass Residuals: ",/)
95          format(2x,"Last Pass Residuals:",/)
            if(ilc .eq. 1) print 90
            if(idone .eq. 1 ) print 95
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      program bayes
c     nonlinear leastsquares algorithm                                c
c                                                                     c
c     This program accomplishes a nonlinear least squares algorithm   c
c     for the problem of estimation of launch vehicle performance     c
c     parameters.  The program uses obser to calculate the Q inverse  c
c     the appropriate H matrix, and the observation matrix. The       c
c     program also uses dhaming to numerically integrate the state,   c
c     and rhs to calculate the EOM and EOV.                           c
c                                                                     c
c     Capt. Dave Vallado            1984                              c
c                                                                     c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

cccccc The common terms

       common /ham/ t,y(72,4),f(72,4),err(72),n,dt,mode
       double precision t,y,f,err,dt
       integer n,mode,nxt

ccccc The other terms

       double precision timeob(500),rho(500),az(500),el(500),
     +        phi(8,8),h(3,8),tmat(3,8),z(3),zpred(3),dx(8,1),
     +        ql(3,3),resid(3),tob,work(8),htql(8,3),pinvo(8,8),
     +        htqlr(8,1),xref(8,1),p(8,8),tepoch,tmatt(8,3),
     +        pinvn(8,8),xrefu(8,1),xmxref(8,1),pndx(8,1),
     +        pndxph(8,1),htqlt(8,8),pinvnc(8,8),pc(8,8),pinvol
     +        (8,8),phic(8,8),phiin(8,8),phit(8,8),beta(8,8),
     +        betat(8,8),pinvo2(8,8)
       integer ila,ilb,ilc,ild,ile,ilf,maxit,nob,trop,ilnob,ilnobs

cccccc READ IN INTIAL DATA AND ALL CONTROL PARAMATERS

       print*,'input epoch time'
       read*,tepoch

       print*,'input initial state vector guess, xref'
       read*,xrefu(1,1),xrefu(2,1),xrefu(3,1),xrefu(4,1),xrefu(5,1),
     +       xrefu(6,1),xrefu(7,1),xrefu(8,1)

       print*,'input the max iterations'
       read*,maxit

       print*,'input the rank of p'
       read*,trop

       print*,'input the number of bayes loop iterations'
       read*,ibloop

       print*,'input the number of data points to be read each
     +    least squares run'
       read*,ilnob

       print*,'input beta, for the fading memory '
       read*,beta(1,1),beta(2,2),beta(3,3),beta(4,4),beta(5,5),
     +       beta(6,6),beta(7,7),beta(8,8)

cccccc print out input

10     format(/,20x,"NONLINEAR BAYES FILTER",/,2x,
     +    "initial state vector :",/,2x,4e20.13,/,2x,4e20.13,/,
     + "initial time :  ",f8.6,"  # of data points :  ",i4,/,2x,
     + "max LS iterations : ",i8," # of bayes chunks :  ",i4,/,2x,
     + "max bayes iterations :  ",i4,"  rank of P :  ",i11,/,2x,
     + "Beta matrix = ",8f6.3,)
```

Bayes Filter Algorithm

1. Pick $\bar{x}_{refu}(t_{epoch})$, initial guess for state vector

   Set Q and $\bar{x}_{ref} = \bar{x}_{refu}$
   read all data and store in z

   set $P^{-1}(-) = 0$ $t_{epoch} = 0$

2. For each Bayes iteration

3. For each least squares iteration

   Set $\phi = [I]$: $T^TQ^{-1}T = 0$: $T^TQ^{-1}\bar{r} = 0$

4. For each observation time

   move $\bar{x}_{ref}(t_{epoch})$ to $\bar{x}_{ref}(t_i)$

   haming and rhs do this to $\bar{x}_{ref}$ also ϕ

   calculate predicted data using $\bar{x}_{ref}(t_i)$ - $\bar{z}pred$
   obser does this

   calculate residuals $\bar{r}_i = \bar{z}_i - \bar{z}pred_i$
   calculate H - obser does this
   $T = H\phi$
   sum $\sum T^TQ^{-1}T$

   sum $\sum T^TQ^{-1}\bar{r}$

5. loop back until each data segment is processed

   $P^{-1}(+) = P^{-1}(-) + T^TQ^{-1}T$

   $\delta\bar{x} = P(+)\ (P^{-1}(-)(\bar{x}(-)-\bar{x}ref) + T^TQ^{-1}\bar{r}\ )$

   $\bar{x}_{ref}(t_o) = \bar{x}_{ref}(t_o) - \bar{x}$

   determine convergence (See page 23)

no

loop back for another L.S. iteration

yes

$P^{-1}(-) = P^{-1}(+)$
$\bar{x}_{ref}(t_i) = y(*,nxt)$

$\bar{x}_{refu}(t_i) = \bar{x}_{ref}(t_i)$

$P^{-1}(-) = \phi^{-1T}P^{-1}\phi^{-1}$

$t_{epoch} = t$

$P^{-1}(-) = \beta\ P^{-1}(-)\ \beta^T$

pinvn(8,8)	$P^{-1}(+)$
pinvnc(8,8)	copy of pinvn
pinvol(8,8)	update of pinvo after each least squares run
htqlr(8,1)	$T^T Q^{-1} \bar{r}$
xref(8,1)	state vector which gets updated
xrefu(8,1)	initial state for each bayes loop
p(8,8)	P matrix (covariance)
pc(8,8)	copy of p matrix
tepoch	Initial start time
tmatt(8,3)	T^T
xmxref(8,1)	$\bar{x}$refu - $\bar{x}$ref

Counters and misc holders
ilb, ilj, ibi, ibg, ibe, ibf, ibc, ibr, ibv, ibw, ilc, ild, ile, ilf, ilg, ilh, ili, ilk, ilm, iln, ilo, ilp, ilq, ilr, ils, ilt, ilv, ilu, pndx(8,1), pndxph(8,1)

Subroutines used

meql
mmpy
mtrans } (See Appendix H)
mprint
eigen
dhaming (See Appendix B)
rhslb (See Appendix C)
obser
razel } (See Appendix G)
randv
mag
cross } (See Appendix A)
radst
lstime

MEQL

This subroutine makes a copy of a matrix. This was used in the Bayes filter program.

APPENDIX I

PROGRAM BAYES.F

Description

This program mechanizes the Bayes filter discussion in chapter 3 and incorporates the summary that is shown on the following page. It reads the truth model data, proceeds to input user data, and then it calculates the state vector as each segment of data is processed. The program is very similar in operation to the lstsq.f program, with the major differences discussed in chapter 3.

User Inputs

tepoch	initial start time
xrefu(8)	initial state vector guess
maxit	max number of least sqaures iterations
trop	rank of P matrix
ibloop	max number of bayes loop iterations
ilnob	number of data points read, each least squares iteration
beta	Weigthing matrix for covariance matrix

Variables to be set

The haming common

Remainging variables

timeob( )	time, rho, az and el for the storage
rho( )	of the truth model data
az( )	
el( )	
phi(8,8)	ϕ
phiin(8,8)	ϕ^{-1}
phit(8,8)	ϕ^T
phic(8,8)	copy of ϕ matrix
h(3,8)	H matrix
tmat(3,8)	T matrix
tob	holds time of each observation
z(3)	holds G matrix values, each observation
zpred(3)	holds G matrix values, calculated from $\bar{x}$ref in the program
dx(8,1)	state vector corrections
q1(3,3)	Q^{-1}
resid(3)	residual vector
work	storage for IMSL inverse routine
htq1(8,3)	$T^TQ^{-1}T$
pinvo(8.8)	$P^{-1}(-)$

```
        do 2020 ilae= 1,3
            stigp(ilae,1)= real(wp(ilae))
2020      continue
        print*,'note that these values are all in random order'
        print*,'eigenvalues equal for the covariance matrix'
        call matprt(stig,8,1)
        print*,'eigenvectors equal for the covariance matrix'
        call matprt(stigv,8,8)

cccccc  now calculate error ellipsoid axis lengths, do conversions

        do 2040 ilad=1,8
            stiger(ilad,1)= dsqrt(stig(ilad,1))
2040      continue

        do 2060 ilaf=1,3
            stiper(ilaf,1)= (dsqrt(stigp(ilaf,1)))*6378.145d+00
            stiger(ilaf,1)= stiger(ilaf,1)*6378.145d+00
2060      continue

        do 2080 ilag=4,6
            stiger(ilag,1)= stiger(ilag,1)*7.90563828d+00
2080      continue
        print*,'the axis lengths for the covariance matrix are'
        call matprt(stiger,8,1)
        print*,'the axis lengths for the position components are'
        call matprt(stiper,3,1)
        return
        end

      include '/en/en84d/dvallado/obser'
      include '/en/en84d/dvallado/dhaming'
      include '/en/en84d/dvallado/rhslb'
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c       this subroutine forms the transpose of a matrix                c
        subroutine mtrans(mat4,mat4r,mat4c,mat5)

        double precision mat4(mat4r,mat4c),mat5(mat4c,mat4r)
        integer mat4r,mat4c

        integer imd,ime

        do 4020 imd=1,mat4r
            do 4020 ime= 1,mat4c
                mat5(ime,imd)= mat4(imd,ime)
4020      continue
        return
        end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c       this subroutine prints a matrix                                c
        subroutine matprt(mat6,mat6r,mat6c)

        double precision mat6(mat6r,mat6c)
        integer mat6r,mat6c

        integer imh,imi

4040    format(10(1x,e12.6))
        do 4030 imh=1,mat6r
            write(*,4040) (mat6(imh,imi),imi=1,mat6c)
4030      continue

         return
          end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                      c
c     this subroutine calculates the eigenvalues and eigenvectors      c
        subroutine eigen(p,trop)

        double precision p(8,8)
        integer trop

        double precision works(16),ponly(3,3),stig(8,1),stigv(8,8),
     +            stiger(8,1),stiper(3,1),stigp(3,1),worksp(6)
        double complex w(8),zeig(8,8),wp(3),zeigp(3,3)
        integer ila,ilaa,ilab,ilag,ilaf,ilad,ilae,ilac

cccccc  calculate eigenvalues and eigenvectors
cccccc  get position only components

        do 1080 ilab=1,3
            do 1080 ilac=1,3
1080            ponly(ilab,ilac)= p(ilab,ilac)

        call eigrf(p,trop,trop,1,w,zeig,16,works,ier)

        call eigrf(ponly,3,3,1,wp,zeigp,6,worksp,ier)

cccccc  transform from complex values to real values

        do 2000 ila=1,8
            stig(ila,1)= real(w(ila))
            do 2000 ilaa=1,8
                stigv(ilaa,ila)= real(zeig(ilaa,ila))
2000      continue
```

```
820        format(/,2x,"current reference trajectory state vector at
     +           at epoch:",/,2x,4e20.13,/,2x,4e20.13,/)
           print 820,xref

cccccc     SUCCESS ?????????
cccccc     check convergence

           ifail = 0
           do 700 ilu = 1,8
              if( dabs(dx(ilu,1)).gt.0.1*dsqrt(dabs(p(ilu,ilu))))
     +                 ifail = 1
700         continue

           if(ifail .eq. 0 ) idone = 1

c-------------------- LOOP BACK FOR NEXT ITERATION -------------------c

9999     continue

cccccc   FAILURE !!!!!!!!!!!

900      format(2x,"maximum iteration limit exceeded
     +          without convergence.")
         print 900
         stop

cccccc    SUCCESS !!!!!!!!!!!!!

5000     continue

850        format(/,2x,"CONVERGENCE ACHIEVED.",/
     +          ,2x,"In nominia Gaussiam trajectorum referentia",/,
     +           2x,"declarium est estimatia",/)
         print 820

cccccc print covariance matrix

950      format(/,2x,"Covariance Matrix at epoch is:",/,
     + 8(1x,8e14.7,/) )
         print 950,p

         call eigen(p,trop)

         end

cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c                                                                    c
c        this program multiplies 2 matricies together                c
         subroutine mmpy(mat1,mat1r,mat1c,mat2,mat2c,mat3)

         double precision mat1(mat1r,mat1c),mat2(mat1c,mat2c),
     +          mat3(mat1r,mat2c)
          integer ima,imb,imc,mat1r,mat1c,mat2c

         do 4000 ima = 1,mat1r
                do 4000 imb = 1,mat2c
                   mat3(ima,imb) = 0.00d+00
                   do 4000 imc = 1,mat1c
                      mat3(ima,imb) = mat3(ima,imb) +
     +                                mat1(ima,imc) * mat2(imc,imb)
4000        continue
         return
         end
```

```
        if( iii .lt. 10) go to 200
        if(( idone .eq. 1).and.(iii.lt.10)) go to 200
        if(( idone .eq. 1).and.(iii.ge.10)) go to 240
        go to 250
200     continue
        print*,'time, res =',tob,(resid(ilm),ilm=1,ndata)

cccccc  if this is last pass, weve already converged,
cccccc        so skip matrix calculations

240     if( idone .eq. 1 ) go to 9000
250     continue

cccccc  extract phi matrix in normal form

        do 105 iln = 1,8
            do 106 ilo = 1,8
106             phi(iln,ilo) = y(8*ilo+iln,nxt)
105       continue

cccccc  form matrix ******  tmat = h * phi

        call mmpy(h,3,8,phi,8,tmat)

cccccc  form matrix ******  htql = T transpose * Q inverse

        call mtrans(tmat,3,8,tmatt)
        call mmpy(tmatt,8,3,ql,3,htql)

cccccc  form matrix ****** pinv = T transpose Q inverse  T
cccccc    (sum through the observations)

        do 140 ilp = 1,8
            do 140 ilq = 1,8
                do 130 ilr = 1,ndata
130                 pinv(ilp,ilq)= pinv(ilp,ilq)+htql(ilp,ilr)
     +                          *tmat(ilr,ilq)
140       continue

cccccc  form matrix ****** htqlr = T transpose Q inverse r
cccccc    (sum through the observations)

        do 150 ils = 1,8
            do 150 ilt = 1,ndata
150             htqlr(ils,1)= htqlr(ils,1)+htql(ils,ilt)*
     +                          resid(ilt)

c------------------ LOOP BACK FOR OBSERVATION LOOP ------------------c

9000      continue
1000    continue

cccccc  have we just finshed printing last pass residuals ?

        if( idone .eq. 1) go to 5000

cccccc  data is processed....improve estimate
cccccc  invert matrix H transpose Q inverse H to find
cccccc  covariance P

        call linv1f(pinv,trop,8,p,0,work,ier)

cccccc  from matrix ******  dx = P * T transpose Q inverse r

        call mmpy(p,8,8,htqlr,1,dx)

cccccc  add in state corrections..

        do 800 ilv = 1,8
800         xref(ilv) = xref(ilv) + dx(ilv,1)

cccccc  print iteration, and current guess

720     format(/,2x,"iteration ",i3,/,2x,"state corrections"
     +        ,/,2x,4e20.13,/,2x,4e20.13 )
        print 720,ilc,dx
```

```
c--------- BEGIN ITERATION LOOP - NONLINEAR LEAST SQUARES ------------c

        dt = timeob(2) - timeob(1)
        do 9999 ilc = 1,maxit

cccccc      REINITIALIZE NUMERICAL INTEGRATION PARAMETERS

            t = tepoch
            mode = 1
            n = 72
cccccc      ics are new reference traj guess

            do 5Ø ild = 1,8
                y(ild,1) = xref(ild)
5Ø            continue

cccccc      phi initial conditions

            do 51 ile = 9,72
51              y(ile,1) = Ø.Ød+ØØ
            do 52 ilf = 9,72,9
52              y(ilf,1) = 1.Ød+ØØ

cccccc      initialize haming and reset the time

            nxt = Ø
            call haming(nxt)
            t= tepoch

cccccc      INITIALIZE BUFFERS FOR MATRIX PRODUCT ACCUMULATION

            do 6Ø ilg = 1,8
                htqlr(ilg,1) = Ø.Ød+ØØ
                do 6Ø ilh = 1,8
6Ø                  pinv(ilg,ilh) = Ø.ØØd+ØØ

cccccc      print first or last pass redidual headers when necessary

63          format(2x,"First Pass Residuals: ",/)
64          format(2x,"Last Pass Residuals:",/)
            if(ilc .eq. 1) print 63
            if(idone .eq. 1 ) print 64

c---------------- OBSERVATION PROCESSING LOOP ------------------------c

            do 13ØØ ili = 1,nob

cccccc          extract each observation

                tob = timeob(ili)
                z(1)= rho(ili)
                z(2)= az(ili)
                z(3)= el(ili)

cccccc          NUMERICALLY INTEGRATE STATE AND PHI TO OBS TIME
c               the number of steps here is equal to 1 since we
cccccc          have dt set exactely the same as the truth data we read

                nstp= 1
                do 8Ø ilk = 1,nstp
8Ø                  call haming(nxt)

cccccc          OBTAIN MATRICES FOR THIS OBSERVATION

                call obser(tob,ql,zpred,h,nxt)

cccccc          MATRIX STUFF - THIS OBSERVATION

                do 1ØØ ill = 1,ndata
1ØØ                 resid(ill) = z(ill) - zpred(ill)

cccccc          selectively print out the iteration data
```

```
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

      program lstsq

c     nonlinear leastsquares algorithm                                 c
c                                                                      c
c     This program accomplishes a nonlinear least squares algorithm    c
c     for the problem of estimation of launch vehicle performance      c
c     parameters.  The program uses obser to calculate the Q inverse   c
c     the appropriate H matrix, and the observation matrix. The        c
c     program also uses dhaming to numerically integrate the state,    c
c     and rhs to calculate the EOM and EOV.                            c
c                                                                      c
c     Capt. Dave Vallado           1984                                c
c                                                                      c
cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

cccccc The common terms

        common /ham/ t,y(72,4),f(72,4),err(72),n,dt,mode
        double precision t,y,f,err,dt
        integer n,mode,nxt

ccccc The other terms

        double precision timeob(5ØØ),rho(5ØØ),az(5ØØ),el(5ØØ),
     +         phi(8,8),h(3,8),tmat(3,8),z(3),zpred(3),dx(8,1),
     +         ql(3,3),resid(3),tob,work(8),htql(8,3),pinv(8,8),
     +         htqlr(8,1),xref(8),p(8,8),tepoch,tmatt(3,3)
        integer ila,ilb,ilc,ild,ile,ilf,maxit,nob,trop

cccccc READ IN INITIAL GUESSES FOR STATE VECTOR, CONTROL PARAM

        print*,'input epoch time'
        read*,tepoch

        print*,'input initial state vector guess, xref'
        read*,xref(1),xref(2),xref(3),xref(4),xref(5),
     +       xref(6),xref(7),xref(8)

        print*,'input the max iterations'
        read*,maxit

        print*,'input the rank of p'
        read*,trop

cccccc print out input

6       format(/,2x,"NONLINEAR LEAST SQUARES",/,2x,
     +       "epoch time :",e2Ø.13,/,2x,
     +       "initial state vector:",/,2x,4e2Ø.13,/,2x,4e2Ø.13,/
     +       , 2x,"maximum iterations:",i3,"  rank of p=",i1/)
        print 6,tepoch,xref,maxit,trop

cccccc    READ IN OBSERVATIONS

        open(unit=14,file='tdata',access='sequential',status='old')
        rewind(unit=14)
        print*,'input the number of observations to be read'
        read*,nob
        do 3Ø ilb = 1,nob
           read (14,*,end=3Ø) rho(ilb),az(ilb),el(ilb),timeob(ilb)
3Ø        continue
        endfile(unit=14)
        ndata= 3

cccccc set last iteration flag

        idone = Ø
```

APPENDIX J

ICBM Test Results 100 Data Point Case

```
initial state vector :
          x                        y                          z                        xdot
 -.1326113091290e+00 -.5769695352120e+00  .8059282248600e+00 -.1025944089500e-02
      ydot                     zdot                       Ve                        M
 -.4449136538300e-02  .6276833691300e-02  .3172982551730e+00  .4479637558632e+01
initial time :        .002000 sec    # of data points :        400
max LS iterations :          8      # of each bayes chunk :  100
max bayes iterations :       3      rank of P :                8
Beta matrix =  1.000 1.000 1.000 1.000 1.000 1.000  .500  .500
Titan-IIIB   launched from 43 N, 132 E 1 DU elevation
Radar site at 52.6 N, 174.1 E, 1 DU elevation

First Pass Residuals:

time, res =  .249577852e-02  .209186890e-08 -.204608340e-06  .275503849e-06
time, res =  .299155705e-02  .611410787e-09 -.207843734e-06  .275112060e-06
time, res =  .348733557e-02 -.120013763e-08 -.209707052e-06  .274805515e-06
time, res =  .398311409e-02 -.342912500e-08 -.209561260e-06  .274554077e-06
time, res =  .447889262e-02 -.616290148e-08 -.216663032e-06  .274406923e-06

iteration   1
state corrections
   .7524464312042e-08 -.1468811488669e-08  .5692291303397e-07  .3335708465224e-05
   .2730453261049e-07 -.3186857906561e-07  .2176948652099e-04 -.2843510440430e-03

current reference trajectory state vector at etime 1:
 -.1326113016045e+00 -.5769695366808e+00  .8059282317829e+00 -.1022608381035e-02
 -.4449109233767e-02  .6276801822721e-02  .3173200246595e+00  .4479352707588e+01

time, res =  .249577852e-02  .751060410e-08  .110439458e-08 -.571607390e-09
time, res =  .299155705e-02  .788293635e-08  .392187394e-10 -.461485322e-09
time, res =  .348733557e-02  .809102872e-08  .457333726e-09 -.358729600e-09
time, res =  .398311409e-02  .805920647e-08  .308801629e-08 -.298545372e-09
time, res =  .447889262e-02  .771118528e-08 -.131271349e-08 -.237021385e-09

iteration   2
state corrections
 -.5665596635395e-08  .1099084931790e-08  .2988739891713e-09  .1067280853176e-07
 -.2792618244811e-07  .3405576184837e-07 -.2288203826774e-04  .2996219419247e-03

current reference trajectory state vector at time 1:
 -.1326113072701e+00 -.5769695355817e+00  .8059282320818e+00 -.1022597708226e-02
 -.4449137159950e-02  .6276835878483e-02  .3172971426213e+00  .4479652329530e+01

time, res =  .249577852e-02  .173993853e-08  .111187259e-08  .521426131e-10
time, res =  .299155705e-02  .212014891e-08  .473456829e-10  .479431581e-10
time, res =  .348733557e-02  .233710567e-08  .466052752e-09  .450908946e-10
time, res =  .398311409e-02  .231518487e-08  .309726567e-08  .804860865e-11
time, res =  .447889262e-02  .197815492e-08 -.130300015e-08 -.196081641e-10

iteration   3
state corrections
   .1918458332356e-12 -.3375004459828e-13 -.1479374160914e-13 -.5513572843075e-12
 -.2165436884948e-12  .4637224828259e-12  .1079067081321e-08  .6122699995441e-08

current reference trajectory state vector at   time 1:
 -.1326113072699e+00 -.5769695355818e+00  .8059282820818e+00 -.1022597708778e-02
 -.4449137160166e-02  .6276835878947e-02  .3172971437003e+00  .4479652335653e+01

Last Pass Residuals:

time, res =  .249577852e-02  .174013353e-08  .111137204e-08  .521455015e-10
time, res =  .299155705e-02  .212034339e-08  .473452338e-10  .479293376e-10
time, res =  .348733557e-02  .233729948e-08  .466052419e-09  .450502413e-10
time, res =  .398311409e-02  .231537783e-08  .309726567e-08  .797097111e-11
time, res =  .447889262e-02  .197834686e-08 -.130299982e-08 -.197327623e-10

CONVERGENCE ACHIEVED.
In nominia Gaussiam trajectorum referentia
declarium est estimatia
```

```
NEXT BAYES LOOP
tepoch = time 2

 First Pass Residuals:

time, res =  .520736309e-01 -.211587923e-08  .254245136e-08 -.539672231e-10
time, res =  .525694094e-01 -.196634043e-08  .237330666e-08 -.115494081e-09
time, res =  .530651879e-01 -.220760424e-08  .173436310e-08 -.513559091e-10
time, res =  .535609665e-01 -.303361597e-08  .398389188e-08 -.496343557e-10
time, res =  .540567450e-01 -.264041814e-08  .250295296e-08 -.138615302e-09

 iteration   1
 state corrections
   .477239446346 7e-05 -.108910711516 9e-05  .303802182476 6e-07 -.732186699079 8e-04
   .806479239515 9e-04 -.102911329815 7e-03  .178622410600 6e-01 -.167613695853 6e+00

 current reference trajectory state vector atat time 2:
  -.132739336129 3e+00 -.577548620233 2e+00  .806816546705 6e+00 -.465063905059 5e-02
  -.198337989374 2e-01  .328514772223 90e-01  .335159334760 4e+00  .431203863979 9e+01

time, res =  .520736309e-01  .483175390e-05  .194892165e-06 -.238352329e-05
time, res =  .525694094e-01  .479114335e-05  .163652054e-06 -.203051882e-05
time, res =  .530651879e-01  .474885516e-05  .132988260e-06 -.169085454e-05
time, res =  .535609665e-01  .470468343e-05  .106250648e-06 -.136435899e-05
time, res =  .540567450e-01  .466042017e-05  .768105582e-07 -.105069666e-05

 iteration   2
 state corrections
   .761521931361 1e-06 -.133482775704 2e-06 -.586188621913 6e-07 -.140062053362 0e-04
   .255666182849 6e-05  .914533103177 4e-06  .949404555752 8e-03 -.203797262656 7e-02

 current reference trajectory state vector a t time 2:
  -.132738624607 3e+00 -.577540753716 0e+00  .806816438086 8e+00 -.466464525593 2e-02
  -.198312422755 9e-01  .328523917570 1e-01  .336108789316 1e+00  .431000066717 2e+01

time, res =  .520736309e-01  .559989979e-05  .195332241e-06 -.237775035e-05
time, res =  .525694094e-01  .555182063e-05  .164775805e-06 -.203369292e-05
time, res =  .530651879e-01  .550179592e-05  .135278702e-06 -.170892141e-05
time, res =  .535609665e-01  .544961655e-05  .110197235e-06 -.140327332e-05
time, res =  .540567450e-01  .539707127e-05  .829090652e-07 -.111642211e-05

 iteration   3
 state corrections
   .815093717609 2e-08 -.141047867893 7e-08 -.666909635376 1e-09 -.137901488898 2e-06
   .187100277068 6e-07  .200453043090 5e-07 -.241885047731 6e-05  .290734964799 9e-04

 current reference trajectory state vector at e time 2:
  -.132738616456 4e+00 -.577540755126 4e+00  .806816407419 9e-00 -.466478315742 0e-02
  -.198312235655 6e-01  .328524118023 1e-01  .336106370465 7e-00  .431002974066 9e+01

time, res =  .520736309e-01  .560812816e-05  .195284734e-06 -.237744218e-05
time, res =  .525694094e-01  .555997694e-05  .164732818e-06 -.203342796e-05
time, res =  .530651879e-01  .550987841e-05  .135240531e-06 -.170870285e-05
time, res =  .535609665e-01  .545762345e-05  .110164184e-06 -.140310438e-05
time, res =  .540567450e-01  .540500077e-05  .828814448e-07 -.111630609e-05

 iteration   4
 state corrections
  -.283438254579 0e-09  .496647309126 5e-10  .218519177347 0e-10  .530684489377 4e-08
  -.900584432502 6e-09 -.453729503861 9e-09  .424877754328 4e-07 -.446526757996 6e-06

 current reference trajectory state vector at epoch:
  -.132738616739 8e+00 -.577540755076 3e+00  .806816407441 7e-00 -.466477785057 6e-02
  -.198312234661 4e-01  .328524113485 8e-01  .336106412953 4e-00  .431002929414 2e+01

 Last Pass Residuals:

time, res =  .520736309e-01  .560784228e-05  .195284667e-06 -.237744529e-05
time, res =  .525694094e-01  .555969384e-05  .164732734e-06 -.203343071e-05
time, res =  .530651879e-01  .550959815e-05  .135240432e-06 -.170870526e-05
time, res =  .535609665e-01  .545734610e-05  .110164071e-06 -.140310647e-05
time, res =  .540567450e-01  .540472639e-05  .828813184e-07 -.111630787e-05

 CONVERGENCE ACHIEVED.
 In nominia Gaussiam trajectorum referentia

 declarium est estimatia
```

```
NEXT BAYES LOOP
tepoch = time 3

 First Pass Residuals:

time, res =  .101651483e+00 -.100673452e-04  .332567295e-05 -.185167969e-04
time, res =  .102147262e+00 -.104727284e-04  .430990151e-05 -.241225845e-04
time, res =  .102643040e+00 -.108936810e-04  .541680948e-05 -.303255215e-04
time, res =  .103138819e+00 -.113308403e-04  .662202038e-05 -.371336172e-04
time, res =  .103634597e+00 -.117838512e-04  .794113728e-05 -.445512757e-04

 iteration   1
 state corrections
   .1408942035930e-05  .1019466118635e-05 -.2264490910225e-05  .7804349800153e-03
   .5697732345062e-03 -.1476070141468e-02  .1468127904633e-01 -.5058682625640e+00

 current reference trajectory state vector at time 3:
  -.1330975613393e+00 -.5790484145171e+00  .8096544898475e+00 -.9459250899923e-02
  -.4135244386006e-01  .8619843067786e-01  .3507876919998e+00  .3804161031578e+01

time, res =  .101651483e+00 -.823612411e-05  .780864914e-06  .966100091e-06
time, res =  .102147262e+00 -.821767890e-05  .779175096e-06  .768552275e-06
time, res =  .102643040e+00 -.819930637e-05  .796032783e-06  .549569884e-06
time, res =  .103138819e+00 -.818143254e-05  .805838625e-06  .305388616e-06
time, res =  .103634597e+00 -.816348789e-05  .823014784e-06  .358946038e-07

 iteration   2
 state corrections
   .7899607587070e-05 -.1527531318060e-05 -.6699753658038e-06 -.3336368992457e-04
   .3822512040808e-04 -.4015400423896e-04  .3521081280592e-01 -.2515317166610e+00

 current reference trajectory state vector at time 3:
  -.1330896617317e+00 -.5790499420484e+00  .8096538198721e+00 -.9492614589848e-02
  -.4131422373966e-01  .8615827667362e-01  .3859985048057e+00  .3552629314917e+01

time, res =  .101651483e+00 -.182580406e-06  .181704796e-08  .962569567e-07
time, res =  .102147262e+00 -.180872879e-06 -.829302715e-08  .108306370e-06
time, res =  .102643040e+00 -.178641234e-06 -.735337102e-08  .142274295e-06
time, res =  .103138819e+00 -.176285749e-06 -.211995642e-07  .195508296e-06
time, res =  .103634597e+00 -.173210684e-06 -.356493463e-07  .269018163e-06

 iteration   3
 state corrections
   .1785326751436e-06 -.3962636388119e-07 -.2576102803298e-08 -.2332915298275e-05
   .3150244566652e-05 -.4974173782535e-05  .6839410314836e-02 -.2533005817415e-01

 current reference trajectory state vector at time 3:
  -.1330894831990e+00 -.5790499816747e+00  .8096538172960e+00 -.9494947505146e-02
  -.4131107349509e-01  .8615330249934e-01  .3928379151205e+00  .3527299256743e+01

time, res =  .101651483e+00 -.184599170e-08  .383802701e-08  .170003333e-08
time, res =  .102147262e+00 -.188518304e-08 -.357831602e-08  .107068482e-08
time, res =  .102643040e+00 -.178554985e-08  .360917363e-08  .273275281e-08
time, res =  .103138819e+00 -.194957889e-08 -.425014024e-09  .408342949e-08
time, res =  .103634597e+00 -.178375517e-08 -.149100083e-08  .618284950e-08

 iteration   4
 state corrections
   .2441997527549e-08 -.5038107129709e-09 -.1332168705860e-09 -.2425232470404e-07
   .2674563279695e-07 -.3642365757096e-07  .6585595590269e-04 -.3034730539397e-04

 current reference trajectory state vector at time 3:
  -.1330894307570e+00 -.5790499821735e+00  .8096538171628e+00 -.9494971757471e-02
  -.4131104674941e-01  .8615326607618e-01  .3929037710764e+00  .3527268909438e+01

time, res =  .101651483e+00  .630525410e-09  .373884612e-08  .803733154e-09
time, res =  .102147262e+00  .565372294e-09 -.356359153e-08 -.413223567e-09
time, res =  .102643040e+00  .629875565e-09  .382533771e-08  .183410289e-09
time, res =  .103138819e+00  .421454777e-09  .792317323e-10 -.102279045e-10
time, res =  .103634597e+00  .533530029e-09 -.610926421e-09  .649873827e-10
```

```
iteration    5
state corrections
   .2316313941378e-11 -.5133465718550e-12 -.2858123034134e-12 -.3341243477990e-10
   .8633814355239e-11  .4240745285131e-11 -.4860173745442e-09  .7662202059526e-08

current reference trajectory state vector at time 3:
 -.1330894307542e+00 -.5790499821791e+00  .8096533171625e-00 -.9494971790884e-02
 -.4131104674077e-01  .8615326608042e-01  .3929037705904e-00  .3527268917100e+01

Last Pass Residuals:


time, res =  .101651483e+00  .633384685e-09  .373855746e-08  .803691522e-09
time, res =  .102147262e+00  .568213934e-09 -.356386831e-08 -.413276981e-09
time, res =  .102643040e+00  .632699247e-09  .382507415e-08  .183333540e-09
time, res =  .103138819e+00  .424260183e-09  .789824872e-10 -.103245433e-10
time, res =  .103634597e+00  .536316831e-09 -.611159900e-09  .648592881e-10

CONVERGENCE ACHIEVED.
In nominia Gaussiam trajectorum referentia
declarium est estimatia
```

Covariance Matrix at the epoch is:

.1738143e-13	-.3129958e-14	-.1227658e-14	-.2334986e-13	.2413318e-14	.4043390e-14	-.5468354e-11	.7542664e-10
-.3129958e-14	.5638242e-15	.2208759e-15	.4188867e-14	-.4554350e-15	-.6951425e-15	.9776474e-12	-.1349248e-10
-.1227658e-14	.2208759e-15	.8702485e-16	.1669509e-14	-.1425370e-15	-.3296298e-15	.3954187e-12	-.5444151e-11
-.2334986e-13	.4188867e-14	.1669509e-14	.7122967e-13	-.5633855e-14	-.1466944e-13	.1677109e-10	-.2312233e-09
.2413318e-14	-.4554350e-15	-.1425370e-15	-.5633855e-14	.4439002e-14	-.4210132e-14	-.5698997e-12	.9363843e-11
.4043390e-14	-.6951425e-15	-.3296298e-15	-.1466944e-13	-.4210132e-14	.1026549e-13	-.4472067e-11	.5962807e-10
-.5468354e-11	.9776474e-12	.3954187e-12	.1677109e-10	-.5698997e-12	-.4472067e-11	.4136680e-08	-.5670804e-07
.7542664e-10	-.1349248e-10	-.5444151e-11	-.2312233e-09	.9363843e-11	.5962807e-10	-.5670804e-07	.7779657e-06

the axis lengths for the covariance matrix are
.553892e-01
.561744e-02
.561744e-02
.000000e+00
.888649e-06
.000000e+00
.000000e+00
.000000e+00

the axis lengths for the position components are
.856473e-03
.144556e-05
.431656e-05

Covariance Matrix at the epoch is:

.2552859e-13	-.4500254e-14	-.1925353e-14	-.2949821e-12	.5475868e-13	.1755796e-13	-.1638970e-11	.1860345e-10
-.4500254e-14	.7934960e-15	.3392427e-15	.5199869e-13	-.9669022e-14	-.3067245e-14	.2863783e-12	-.3258160e-11
-.1925353e-14	.3392427e-15	.1455539e-15	.2226362e-13	-.4105687e-14	-.1370131e-14	.1276282e-12	-.1436683e-11
-.2949821e-12	.5199869e-13	.2226362e-13	.5477386e-11	-.9946998e-12	-.3636196e-12	.3050277e-10	-.3475137e-09
.5475868e-13	-.9669022e-14	-.4105687e-14	-.9946998e-12	.1838933e-12	.6079707e-13	-.5006759e-11	.5862689e-10
.1755796e-13	-.3067245e-14	-.1370131e-14	-.3636196e-12	.6079707e-13	.3262684e-13	-.2880377e-11	.3026758e-10
-.1638970e-11	.2863783e-12	.1276282e-12	.3050277e-10	-.5006759e-11	-.2880377e-11	.2781686e-09	-.2855496e-08
.1860345e-10	-.3258160e-11	-.1436683e-11	-.3475137e-09	.5862689e-10	.3026758e-10	-.2855496e-08	.2987423e-07

the axis lengths for the covariance matrix are
.171632e-01
.000000e+00
.000000e+00
.357203e-05
.357203e-05
.714242e-06
.000000e+00
.000000e+00

the axis lengths for the position components are
.103764e-02
.174160e-05
.426545e-05

Covariance Matrix at the epoch is:

.3033197e-13	-.5046911e-14	-.2799403e-14	-.5702880e-12	.1022399e-12	.3700491e-13	-.1301219e-11	.9428363e-11
-.5046911e-14	.8399119e-15	.4656509e-15	.9489841e-13	-.1702645e-13	-.6134138e-14	.2143572e-12	-.1558434e-11
-.2799403e-14	.4656509e-15	.2587450e-15	.5265183e-13	-.9415312e-14	-.3467499e-14	.1244752e-12	-.8914157e-12
-.5702880e-12	.9489841e-13	.5265183e-13	.1560463e-10	-.2747826e-11	-.1157845e-11	.3576544e-10	-.2609134e-09
.1022399e-12	-.1702645e-13	-.9415312e-14	-.2747826e-11	.4841875e-12	.1971641e-12	-.5801309e-11	.4342816e-10
.3700491e-13	-.6134138e-14	-.3467499e-14	-.1157845e-11	.1971641e-12	.9730505e-13	-.3503794e-11	.2389630e-10
-.1301219e-11	.2143572e-12	.1244752e-12	.3576544e-10	-.5801309e-11	-.3503794e-11	.1981082e-09	-.1170110e-08
.9428363e-11	-.1558434e-11	-.8914157e-12	-.2609134e-09	.4342816e-10	.2389630e-10	-.1170110e-08	.7181020e-08

the axis lengths for the covariance matrix are
.268372e-01
.000000e+00
.362999e-02
.449934e-05
.000000e+00
.861085e-06
.000000e+00
.000000e+00

the axis lengths for the position components are
.113075e-02
.191221e-05
.427634e-05

SUMMARY OF ALL CASES

	V_e (DU/TU)	M	
Exact			
Stage I	.317298255e+00	.447963755e+01	
Stage II	.392904480e+00	.352726545e+01	

Data Segments = 16

Point #:			Converged on iteration #:
16	.317315725e+00	.447938985e+01	2
32	.317193607e+00	.448102999e+01	1
48	.317368164e+00	.447874052e+01	1
64	.317439486e+00	.447790352e+01	1
80	.317345804e+00	.447907412e+01	1
96	.317478537e+00	.447760474e+01	1
112	.317518937e+00	.447727873e+01	1
128	.317471044e+00	.447786183e+01	1
144	.317562390e+00	.447706513e+01	1
160	.317353960e+00	.447912668e+01	1
176	.316971039e+00	.448249374e+01	1
192	.675400244e+00	.269968034e+01	4
208	.473677628e+00	.311701023e+01	3
224	.392931745e+00	.352711260e+01	3
240	.393296229e+00	.352519154e+01	2

Data Segments = = 24

24	.317305053e+00	.447954511e+01	2
48	.317304049e+00	.447955637e+01	1
72	.317283474e+00	.447981709e+01	1
96	.317353812e+00	.447900588e+01	1
120	.317308991e+00	.447952553e+01	1
144	.317392974e+00	.447871385e+01	1
168	.317268608e+00	.447990303e+01	1
192	.361079891e+00	.414022556e+01	3
216	.406305787e+00	.345281348e+01	4
240	.393116024e+00	.352614137e+01	3

Data Segments = 32

32	.317308409e+00	.447949574e+01	2
64	.317306635e+00	.447953176e+01	1
96	.317305226e+00	.447955992e+01	1
128	.317324984e+00	.447935543e+01	1
160	.317285360e+00	.447976097e+01	1
192	.331250475e+00	.436375095e+01	3
224	.396913417e+00	.350494767e+01	4
256	.392841545e+00	.352758616e+01	2

Continued:

Point #:	V_e (DU/TU)	M	Converged on iteration #
	Data Segments = 48		
48	.317303338e+00	.447956672e+01	2
96	.317296296e+00	.447966247e+01	1
144	.317295326e+00	.447966670e+01	1
192	.320304527e+00	.445333709e+01	3
240	.393636292e+00	.352327942e+01	5
	Data segments = 50		
50	.317304110e+00	.447955585e+01	2
100	.317296361e+00	.447966253e+01	1
150	.317291091e+00	.447970933e+01	1
200	.686866862e+00	.261674361e+01	3
250	.392921506e+00	.352717312e+01	5
	Data Segments = 64		
64	.317299985e+00	.447961325e+01	2
128	.317300626e+00	.447960982e+01	1
192	.318331699e+00	.447028787e+01	2
256	.393127634e+00	.352608074e+01	5
320	.392905747e+00	.352725914e+01	2
	Data Segments = 66 *		
66	.317298046e+00	.447964018e+01	2
132	.317302351e+00	.447958587e+01	1
198	.416586494e+00	.356484600e+01	4
254	.392918156e+00	.352720315e+01	4
320	.392871250e+00	.352740128e+01	2
	Data segments = 66 **		
66	.317300432e+00	.447960725e+01	2
132	.317301918e+00	.447959537e+01	1
198	.364060396e+00	.409734550e+01	4
254	.392912142e+00	.352722456e+01	4
320	.392902400e+00	.352727440e+01	2
	Data Segments = 100		
100	.317297143e+00	.447965233e+01	1
200	.336106412e+00	.431002929e+01	4
300	.392903770e+00	.352726891e+01	5

* Values of .9 .9 .9 .9 .9 .9 .7 .7

** Values of .8 .8 .8 .8 .8 .8 .5 .5

VITA

David A. Vallado was born on May 14, 1958 in Winchester Massachusetts. He graduated from Parsippany Hills High School in 1976. He attended the United States Air Force Academy and graduated in 1980. He received a Bachelor of Science in Astronautical Engineering. His first assignment was as the project officer for the M-X Stage I at the Ballistic Missile Office at Norton AFB, California. While stationed at Norton AFB, he also completed a Master of Science in Systems Management from the University of Southern California in 1982. He entered the School of Engineering, Air Force Institute of Technology in June 1983. He met and married his wife Laura while at AFIT.

Permanent address: 3 Glencove Road
Morris Plains, New Jersey 07950

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A151 927

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GA/AA/84D-10	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering — 6b. OFFICE SYMBOL (If applicable) AFIT/AA	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION — 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19	
12. PERSONAL AUTHOR(S) David A. Vallado, M.S., B.S., Capt., USAF	
13a. TYPE OF REPORT MS Thesis — 13b. TIME COVERED FROM ____ TO ____	14. DATE OF REPORT (Yr., Mo., Day) 1984 December — 15. PAGE COUNT 128
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES

FIELD	GROUP	SUB. GR.
19	04	
16	02	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Estimation — Rocket and Missile Trajectory
Bayes Filter — Parameter Identification

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: ESTIMATION OF LAUNCH VEHICLE PERFORMANCE PARAMETERS

Thesis Chairman: Dr. William E. Wiesel

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. William E. Wiesel, Professor	22b. TELEPHONE NUMBER (Include Area Code) 513-255-3517 — 22c. OFFICE SYMBOL AFIT/ENY

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

The estimation of launch vehicle performance parameters was explored through the use of a Bayes Filter. The main emphasis was to use an eight state model that would include the vehicle position and velocity vectors, the vehicle exhaust velocity, and the ratio of the mass flow rate to the initial mass. A primary objective was to be able to observe these quantities through the staging events, where the last two elements would be changing very rapidly. The results indicated that indeed the staging event was observable. However, as would be expected, the data processed at the exact time of staging included errors which diminished as the filter processed more data. A fading memory was added in an attempt to improve the filters performance in the area of a staging event. This proved to be marginally successful as several Bayes loop iterations had to be performed to notice the effect of the fading memory addition. Care was taken to show each step of the filter development and its checkout. Several numerical tables are presented including the input and output data.

END

FILMED

4-85

DTIC